PARANINFO
THOMSON LEARNING

INCLUYE CD-ROM



SISTEMAS DIGITALES

y Tecnología de Computadores

José Mª Angulo Usategui
Javier García Zubía

Para comunicar la ciencia es menester persuadirla,
y persuadirla es hacerla amable;
es necesario despojarla del oscuro científico aparato,
simplificarla,
acomodarla a la comprensión general e inspirarle
aquella fuerza,
aquella gracia que, fijando la imaginación,
cautiva victoriosamente la atención de cuantos la oyen.

Jovellanos

ÍNDICE

Prólogo

La presente obra es fruto del trabajo y la experiencia de sus autores en la formación y enseñanza de la Tecnología Digital a los alumnos del primer curso en diversas especialidades de ingeniería en la Universidad de Deusto. Intenta servir como una eficaz herramienta para la comprensión y el manejo de la moderna Electrónica Digital y de los cimientos en los que se basa la Tecnología de los Computadores. La obra está enfocada especialmente al estudio de la Teoría de Sistemas Digitales y a las bases en las que se fundamenta el funcionamiento y la construcción de computadores. Como complemento a este texto sus creadores recomiendan complementarlo con las prácticas de manejo y diseño con circuitos digitales recogidas en el laboratorio Universal Trainer (www.microcontroladores.com).

El libro se compone de trece capítulos, de los cuales los cuatro primeros se dedican a describir y aplicar los principios de la Tecnología Digital. El primero introduce el concepto de digital, su implementación y todas las aplicaciones que soporta. El siguiente capítulo describe los sistemas de numeración y la aritmética usados en el cómputo digital. El capítulo 3 desarrolla el álgebra de Boole -principal herramienta de análisis y diseño digital- con claridad y profundidad, pero evitando las complicaciones matemáticas y apoyándose en numerosos ejemplos y ejercicios. El capítulo 4, que cierra este primer bloque del libro, explica la construcción y características de los circuitos integrados usados en las aplicaciones.

Sentadas las bases, el resto de la obra aborda el análisis y diseño de sistemas combinacionales y secuenciales. En el primer capítulo de esta parte se detallan las estrategias y métodos de análisis y diseño de sistemas combinacionales, tanto a nivel de bit, como a nivel de palabra o funcional. El capítulo 6 contempla las mismas necesidades que el anterior, pero en el marco particular de los elementos aritméticos. El siguiente capítulo establece los fundamentos de los elementos básicos de memoria -los flip-flops-, que a su vez serán el soporte del resto de la obra. Los capítulos 8 y 9 se centran en dos tipos de circuitos secuenciales en especial: registros y contadores. El análisis y diseño de autómatas se aborda en el capítulo 10 con numerosos ejemplos. El tema específico de las memorias es tratado en el capítulo 11. El capítulo 12 utiliza los conocimientos y métodos obtenidos en los anteriores para construir un computador completo: la Máquina Sencilla. El capítulo 13 y último es el manual de usuario del programa BOOLE-DEUSTO incluido en el CD del libro.

El libro en su orientación busca la claridad y la efectividad. Incluye un buen número de métodos, todos ellos justificados, ordenados y comprobados con ejemplos, lo que asegura su utilidad. Por otra parte, permite al lector acercarse a los distintos bloques según sus necesidades: con detalle o genéricamente, dando de cada bloque su visión externa e interna. Además se ha preferido mantener el espíritu original de claridad y efectividad, aun a costa de no cubrir todos los aspectos de la Electrónica Digital. La estructura y los contenidos se adaptan a la perfección tanto a la Formación Profesional como a los estudios universitarios de ingeniería electrónica e informática, sin descuidar a los autodidactas.

Con objeto de facilitar la comprensión de muchas secciones del texto, viene acompañado por un CD con el siguiente contenido:

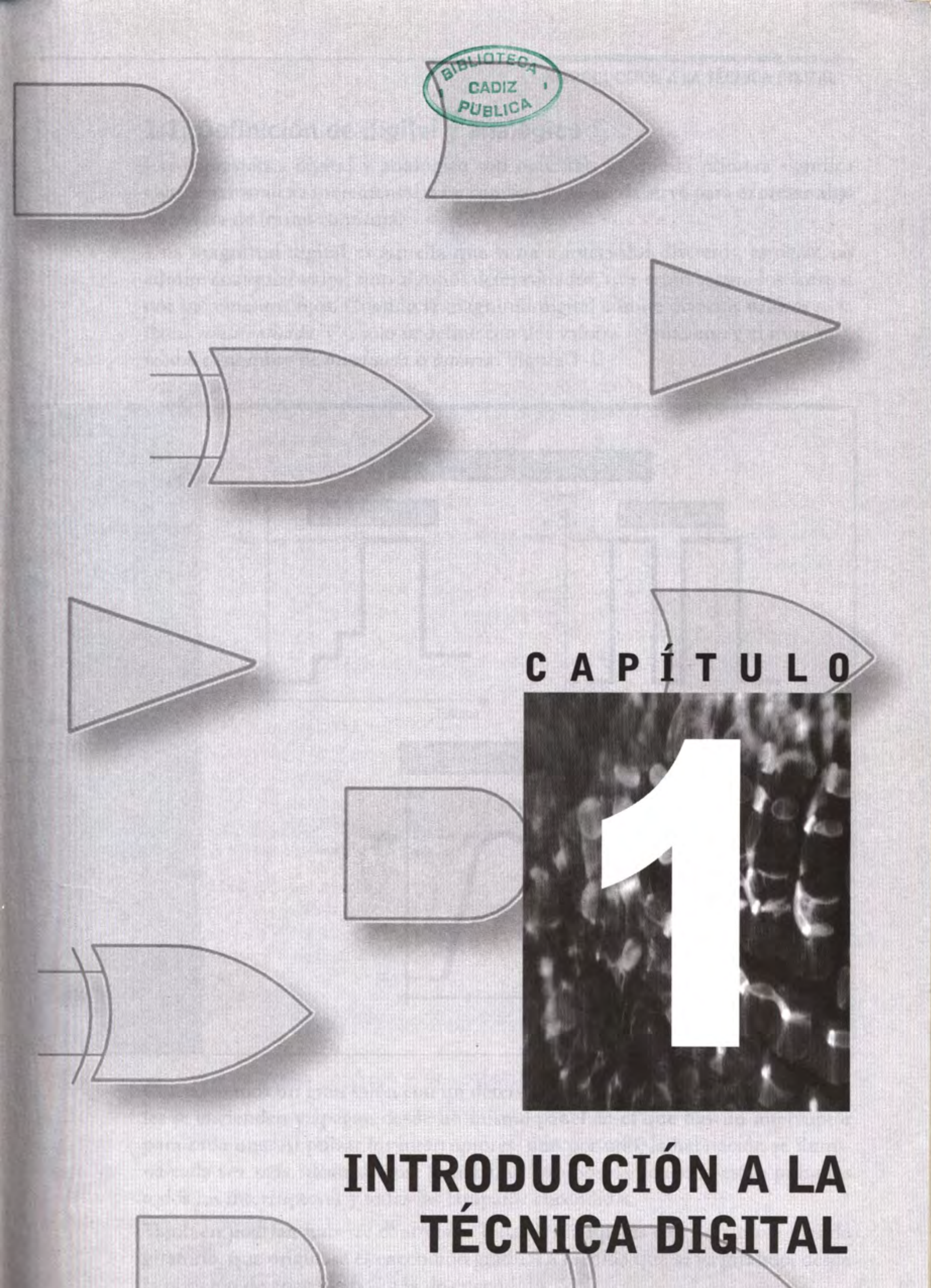
a) Descripción Técnica de Circuitos Integrados Comerciales. En esta sección el lector encontrará las hojas técnicas de algunos de los Circuitos Integrados presentados en el libro.

b) Programa BOOLE-DEUSTO para el análisis y diseño automático de sistemas digitales combinacionales y secuenciales a nivel de bit. Permite capturar y obtener tablas de verdad, expresiones booleanas, diagramas de Veitch-Karnaugh, expresiones simplificadas, diagramas de autómatas de Moore y Mealy, circuitos lógicos, etc. BOOLE permite pasar de unas representaciones a otras con entera libertad, convirtiéndose en la calculadora booleana del curso. BOOLE-DEUSTO ofrece lo anterior desde la sencillez, buscando convertirse en un estándar en la enseñanza de Sistemas Digitales. Avala este propósito el haber adoptado en el desarrollo (más de 5 años y 25.000 líneas de código) el punto de vista del alumno-profesor, y no el del profesional. Este programa está orientado al aula y no a la industria. BOOLE-DEUSTO ha sido presentado en varios congresos, nacionales e internacionales, donde ha conseguido el reconocimiento de profesores de otros centros y un premio. El Congreso TAEE es un foro que reúne cada dos años a los profesionales en la enseñanza de electrónica. En el TAEE 2000, celebrado en Barcelona, BOOLE-DEUSTO recibió el "Premio al Mejor Equipo Software".

c) El programa "Máquina Sencilla", que simula el comportamiento del computador desarrollado en el texto y permite su programación. Este programa ya estuvo incluido en la obra "Introducción a los Computadores" de esta editorial.

Tanto el texto como las herramientas lógicas contenidas en el CD, y las experiencias del laboratorio UNIVERSAL TRAINER, son las tres herramientas que utilizan con enorme éxito los dos autores en la formación integral de sus alumnos en el primer contacto con la Electrónica Digital, y que sirven para forjar a los mejores ingenieros.

BIBLIOTECA
CADIZ
PUBLICA



CAPÍTULO 1

INTRODUCCIÓN A LA TÉCNICA DIGITAL

1.1. Definición de digital y analógico

Las expresiones **digital** y **analógico** son opuestas, ya que la primera significa algo de naturaleza incremental y, en cambio, la segunda sirve para expresar algo que varía de forma continua.

Una **magnitud digital** es aquella que varía a intervalos discretos, es decir, no admite cualquier valor, sino algunos determinados, que están separados entre sí por incrementos fijos. Cuando la magnitud digital admite diversos valores se la llama *multivaluada,* y si sólo se define con dos valores (el máximo y el mínimo), recibe el nombre de *bivaluada* o *binaria*. Figura 1-1.

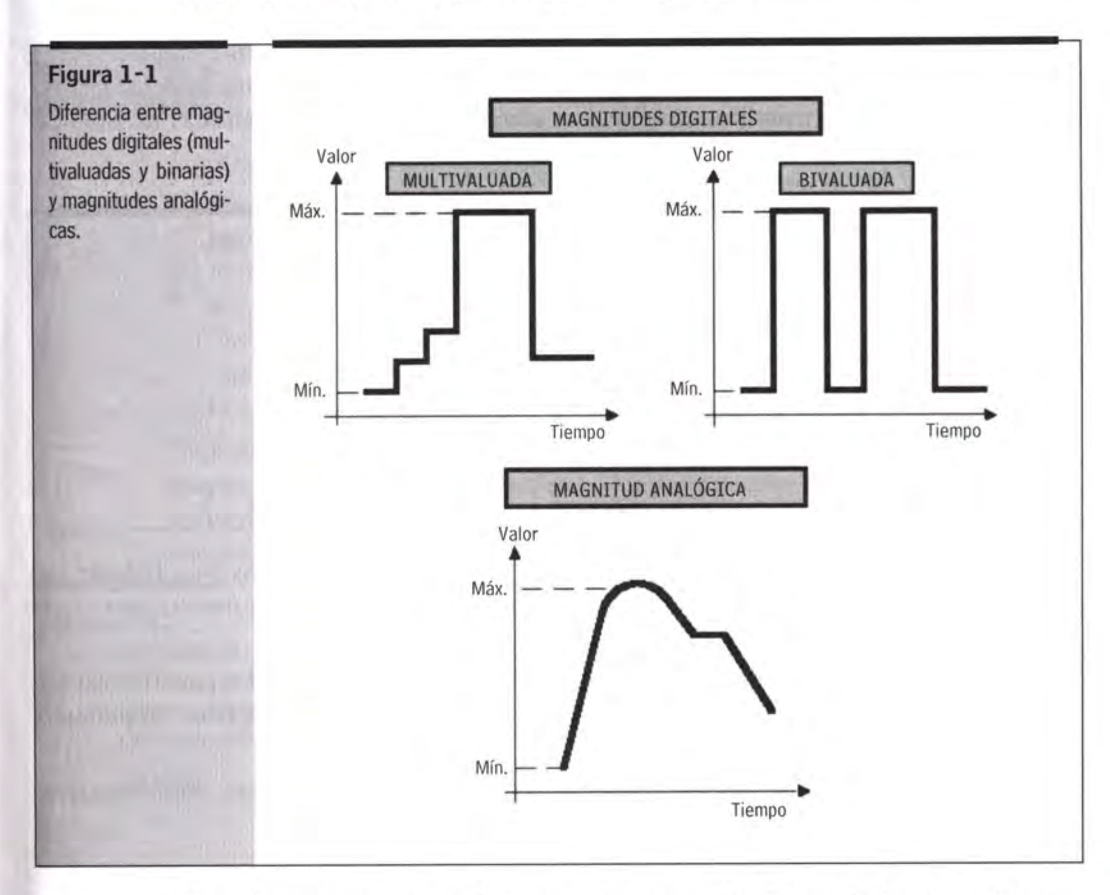


Figura 1-1
Diferencia entre magnitudes digitales (multivaluadas y binarias) y magnitudes analógicas.

Consideremos un gran salón con un determinado número de lámparas, las cuales se encienden y apagan desde un mismo panel en el que hay un interruptor para cada una. Al pulsar los interruptores, uno por uno, la habitación se ilumina cada vez más, alcanzándose la máxima luminosidad cuando están pulsados todos los interruptores y todas las lámparas encendidas.

También podrían haberse controlado todas las lámparas con un simple mando giratorio, que originase el encendido gradual a medida que se va girando, desde la posición de apagado hasta la de encendido.

En el primer caso, la regulación de luminosidad se efectúa mediante incrementos discretos, mientras que en el segundo es de manera continua.

Dos buenos ejemplos (que pueden ser tanto analógicos como digitales) son los relojes y los voltímetros. Las agujas principales de un reloj corriente se mueven continuamente, mientras que en un reloj digital los números cambian, de repente, al final de cada minuto o segundo. Del mismo modo, un voltímetro analógico dispone de una aguja de medida que puede desplazarse gradualmente desde un extremo al otro de la escala, mientras que en un voltímetro digital la tensión se muestra mediante dígitos discretos, que cambian de repente.

- Las magnitudes digitales varían de forma incremental a intervalos discretos, mientras que las magnitudes analógicas lo hacen de forma continua.
- Las magnitudes digitales admiten un número determinado de valores.
- Las magnitudes analógicas admiten infinitos valores entre el máximo y el mínimo.

Ejemplo 1-1

¿Cuáles de los siguientes elementos funcionan con magnitudes digitales?

(a) El pedal del acelerador del coche.
(b) El mando para las luces del coche.
(c) La llave de puesta en marcha del coche.
(d) El manillar de una moto.
(e) El pedal del freno del automóvil.

SOLUCIÓN

(b) y (c).

Un **sistema digital** es un conjunto de elementos diseñados para trabajar con magnitudes digitales. El sistema digital más importante de nuestra época es el "computador digital".

Un **sistema analógico** es un conjunto de elementos diseñados para trabajar con magnitudes analógicas. Un ejemplo de este tipo de sistemas es un amplificador de sonido.

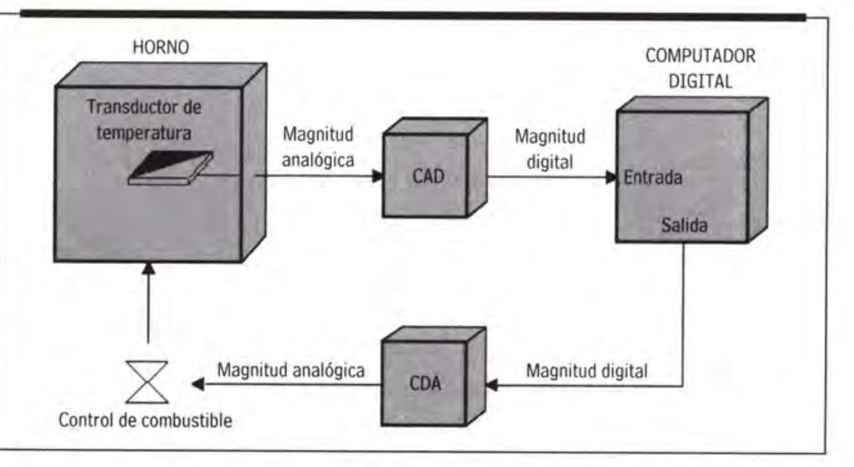
Los elementos que configuran un sistema pueden ser de diversa naturaleza, pero sólo se hace referencia expresa a los de tipo electrónico.

1.2. Ventajas e inconvenientes de los sistemas digitales

La mayoría de las magnitudes físicas de nuestro mundo son de *carácter analógico* (temperatura, iluminación natural, velocidad, presión, etc.). Sin embargo, las máquinas que se emplean para su control son de tipo digital, dadas las ventajas que ofrecen. Así, por ejemplo, el control automático de la temperatura de un horno se realiza mediante un computador digital, aunque la temperatura sea una magnitud analógica. Esto requiere el empleo de etapas **conversoras** que transforman lo analógico a digital (CAD: Conversor Analógico-Digital) y lo digital en analógico (CDA: Conversor Digital-Analógico). Figura 1-2.

Figura 1-2

Las magnitudes y dispositivos típicos en la industria son de carácter analógico. El proceso de la información y los resultados los soporta un computador digital, que requiere un CAD en entrada y un CDA en salida.



Las principales ventajas de los sistemas digitales son:

1ª) Son más sencillos y económicos que los analógicos al tener que manejar sólo unos pocos valores.

2ª) Son más seguros y precisos. La precisión de los sistemas digitales puede ser tan grande como se quiera, añadiendo más elementos.

3ª) Dada la naturaleza discreta de las magnitudes digitales, la información de este tipo es más fácil de almacenar.

4ª) Los circuitos digitales son más resistentes a las interferencias y ruidos externos.

El principal inconveniente de los sistemas digitales proviene del hecho de que la mayoría de las magnitudes físicas, como los controladores y activadores del mundo industrial, son de tipo analógico. Esto supone el encarecimiento y aumento de la complejidad de los diseños ante la necesidad de disponer de CAD y CDA.

Ejemplo 1-2

Un transductor digital de temperatura está construido para medir un mínimo de 10 °C y un máximo de 110 °C. El transductor proporciona 0,5 V en su salida a la temperatura de 10 °C, y 5,5 V a la de 110 °C. Sabiendo que el incremento o escalón entre posibles valores de salida es de 0,5 V, indicar qué temperaturas puede medir y los voltajes correspondientes.

SOLUCIÓN

TEMPERATURA (ºC)	VOLTAJE DE SALIDA
10	0,5
20	1
30	1,5
40	2
50	2,5
60	3
70	3,5
80	4
90	4,5
100	5
110	5,5

1.3. Lógica digital y electrónica digital

El estudio de la **LÓGICA DIGITAL** requiere la consideración de dos aspectos diferentes: **el proceso lógico**, que es la base teórica de los computadores, calculadoras electrónicas, relojes digitales y otros aparatos electrónicos digitales, y **el circuito electrónico**, con el que se construyen todos los dispositivos mencionados. La "*toma de decisiones*" es el objetivo de la lógica digital y el circuito electrónico es quien lo ejecuta o realiza.

El ser humano está familiarizado por naturaleza con la lógica, puesto que su mente está usándola continuamente para la realización de funciones de toma de decisión. Así, podemos resolver problemas matemáticos, tomar decisiones basadas en hechos acontecidos y modificar nuestras decisiones como resultado de nuevas informaciones o con el conocimiento adquirido previamente y almacenado en nuestra memoria. Nuestra mente es una aproximación de lo que la lógica digital lleva a cabo electrónicamente, al menos cuando nuestros aspectos emocionales o intuitivos están completamente superados.

Recibe el nombre de **ELECTRÓNICA DIGITAL** el conjunto de circuitos electrónicos que realizan las operaciones necesarias para obtener las decisiones lógicas. Son significativamente diferentes a las que se usan, por ejemplo, en los receptores de radio y televisión, cuyos circuitos forman parte de la denominada **ELECTRÓNICA ANALÓGICA**.

El estudio de la Electrónica Digital no requiere grandes conocimientos previos de Electrónica, porque tiene una gran semejanza con los procesos racionales del pensamiento en nuestra mente. Nosotros expresamos las decisiones hablando, escribiendo o actuando. Asimismo, las decisiones digitales se expresan mediante señales electrónicas. Aprendiendo a reconocer las características de dichas señales y conociendo las reglas esenciales con las que operan los circuitos lógicos, se comprende lo que es la **Lógica Digital,** no siendo preciso conocer la teoría electrónica de cada circuito individual y el comportamiento de sus elementos discretos, tales como transistores, diodos y resistencias.

Existe una gran relación entre la Lógica Digital y la matemática o la filosófica, lo cual tiene un gran valor a la hora de analizar y usar los circuitos lógicos digitales, pero su origen proviene de los circuitos eléctricos a base de relés, que se usaron mucho antes de conocerse los computadores digitales. Figura 1-3.

Los circuitos electrónicos que maneja la Lógica Digital son más sencillos que los dedicados a trabajar con valores analógicos, que varían entre sus límites pudiendo tomar infinitos valores. El error que se comete en la representación digital (al no poder representar todos los valores) queda compensado por la sencillez y economía de su construcción.

En Electrónica, los parámetros habituales son el **voltaje** y la **corriente**, los cuales son controlados, fundamentalmente, por los transistores. Cuando se trabaja con magnitudes digitales binarias, que sólo representan los dos valores límites, el transistor funciona como un interruptor, es decir, con sólo dos puntos de trabajo: el **punto de *corte***, en el que no deja circular corriente ($I_C = 0$) y que se ase-

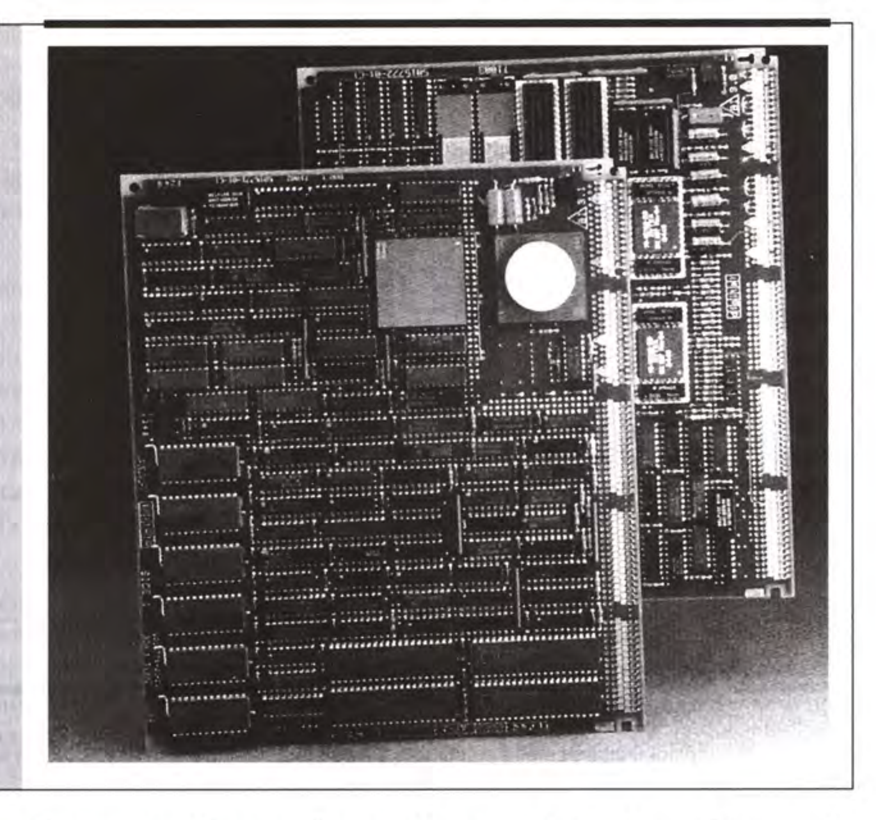


Figura 1-3
Fotografía de un par de tarjetas con circuitos integrados destinadas a la fabricación de computadores.

meja al interruptor abierto, y el **punto de *saturación***, en el que deja pasar la máxima corriente ($I_{C\,máx}$) y que es similar al comportamiento de un interruptor cerrado, que no ofrece resistencia al paso de la corriente. Figura 1-4.

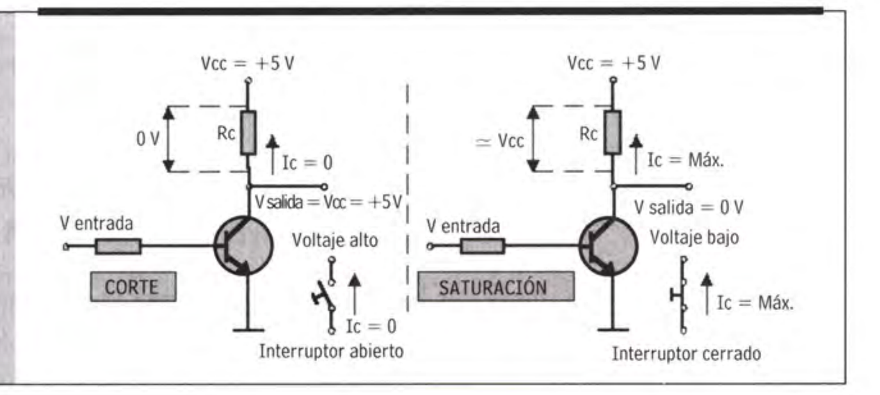


Figura 1-4
El transistor es el elemento fundamental usado en Electrónica Digital. Admite dos puntos de funcionamiento en esta aplicación: corte y saturación.

Cuando el transistor trabaja en el punto de corte, no hay caída de tensión en la resistencia de carga R_C y en la salida la tensión es aproximadamente igual a la V_{cc} de +5 V (**voltaje alto**), mientras que, cuando trabaja en el punto de saturación, R_C absorbe prácticamente toda la V_{cc} con lo que en la salida la tensión es de 0 V (**voltaje bajo**).

Ejemplo 1-3

A) El transistor de la figura 1-5 está casi en corte, pues sólo deja pasar 0,5 mA. Calcular el voltaje de salida.

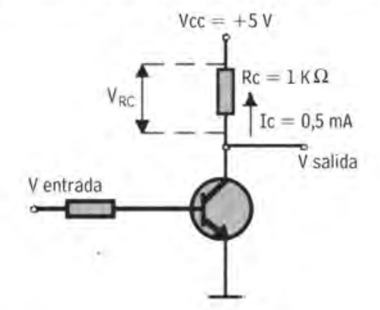


Figura 1-5

Circuito electrónico del transistor en estado de "casi" corte (IC = 0,5 mA).

SOLUCIÓN

El voltaje que absorbe R_C es:

$V_{RC} = R_C \cdot I_C = 1.000 \cdot 0{,}0005 = 0{,}5$ V

El voltaje de salida se obtiene restando de V_{cc} el que absorbe R_C :

$V_{salida} = V_{cc} - V_{RC} = 5 - 0{,}5 = 4{,}5$ V

Ejemplo 1-4

B) La Lógica Digital es similar a la lógica que usamos en la toma de decisiones. Entre las siguientes frases, indica cuáles son "lógicas":

(a) Cuando nieva y hace frío, voy a bañarme a la playa.

(b) Si llueve y salgo a la calle, cojo el paraguas.

(c) Inicio el adelantamiento a otro vehículo cuando veo despejada la carretera.

(d) Cruzo la calle cuando tengo en rojo el semáforo correspondiente.

SOLUCIÓN

(b) y (c).

1.4. Elementos de decisión y memoria

Para entender mejor lo que hace la Lógica Digital, vamos a examinar algunas funciones específicas de la mente humana que pueden encontrar un duplicado en la Lógica Digital.

La función con la que la mente toma decisiones es tal que, si ciertos factores se cumplen o son verdad, como resultado puede decidir que otros factores también se cumplirán.

Por ejemplo, si vemos un semáforo en rojo mientras conducimos un coche, la mente toma la decisión de detenerlo. De una forma elemental, este proceso se puede simular con un circuito electrónico que se denominará "elemento de toma de decisión" o, más concretamente, **puerta lógica**.

Por ejemplo, la puerta lógica puede recibir señales eléctricas y, si ambas alcanzan el voltaje requerido, aparece una tercera señal en la salida. En otras palabras, toma una decisión (salida), que es función del estado de las dos entradas.

Otra cosa que puede hacer la mente humana es tomar una decisión en función de un acontecimiento o factor ya pasado o recordado (MEMORIA).

Por ejemplo, las reglas del ajedrez se deben memorizar para tomar decisiones mientras se juega. Un niño que recuerda la quemadura que tuvo en una mano, la aparta de la estufa caliente. La capacidad de memoria que tiene la mente humana puede compararse con la que tiene la Electrónica Digital, en la cual existe un **elemento de memoria** capaz de recordar por un período de tiempo indefinido una señal de nivel lógico recibida en el pasado. Este elemento puede recordar la existencia de una señal pasada y proporcionarla cuando sea necesario. También es capaz de borrarla y quedar preparada para recibir una nueva señal.

Interconectando muchas puertas o elementos de decisión junto con elementos de memoria, se pueden almacenar muchas señales, que transmiten información codificada, y tomar decisiones muy complejas en cuestión de millonésimas de segundo. Aunque estos circuitos pueden ser muy complicados, al tener muchos elementos, hay maneras sistemáticas y simplificadas para analizarlos. Todos estos circuitos se basan exclusivamente en dos elementos simples: *puertas y memorias*.

Ejemplo 1-5

Señale cuál de las acciones que se indican es la que corresponde al comportamiento de una puerta lógica:

(a) Almacena la información introducida por su entrada un tiempo determinado.

(b) La información de su salida es proporcional a la de sus entradas.

(c) El valor de su salida depende de los valores de sus entradas.

SOLUCIÓN

(c).

1.5. Naturaleza binaria de la lógica digital

Mientras en los circuitos analógicos pueden existir al mismo tiempo muchos voltajes diferentes, en los digitales sólo hay dos. Esto significa que, usando estos dos estados lógicos, puede codificarse cualquier número, letra del alfabeto u otra información. Estos dos voltajes reciben el nombre de "estado lógico 0" y "estado lógico 1", o también, "falso" y "verdadero", y nombres parecidos. Debido al uso de sólo dos estados, se dice que la Lógica Digital es **binaria por naturaleza**.

El significado de esta naturaleza binaria de la Lógica Digital es correcto, puesto que los circuitos lógicos pueden obtener todas sus funciones de decisión y memoria usando dos estados lógicos.

En el funcionamiento de los circuitos de Lógica Digital se utilizan sólo dos estados; por eso se emplea el sistema binario para codificar la información, el cual es tan versátil y útil como cualquier otro, y mucho más fácil para diseñar circuitos digitales. Para entender esto último, se analizan a continuación los sistemas numéricos en general.

Ejemplo 1-6

Indíquese qué elementos de los que se citan tienen naturaleza binaria:

(a) Un interruptor de dos posiciones.
(b) El sintonizador de emisoras de radio.
(c) El sistema de numeración decimal.
(d) El timbre de casa.
(e) El velocímetro del coche.
(f) El piloto indicador del freno de mano activo en los vehículos.

SOLUCIÓN

(a), (d) y (f).

1.6. Sistemas de numeración

Estamos acostumbrados a usar el sistema de numeración decimal y a contar del 1 al 10. Dicho sistema tiene 10 estados básicos o dígitos, desde el 0 hasta el 9. Cuando se quiere contar por encima de nueve, se combinan dos o más de los dígitos básicos y de esta forma podemos codificar números tales como el 10, 100, 1.000 y mucho mayores. No hay razón alguna por la que no pueda usarse un sistema de ocho estados (octal) o de dos (binario).

El sistema binario (con sólo dos estados) es semejante al decimal, excepto en que, para expresar un número, se requieren más dígitos que en el decimal. De esta forma, para expresar los números decimales en el sistema binario se utilizan las siguientes expresiones:

BINARIO		DECIMAL
0000	=	0
0001	=	1
0010	=	2
0011	=	3
0100	=	4
0101	=	5

No hay muchas cosas en la naturaleza que sean múltiplos de 10 a no ser, por ejemplo, los dedos de nuestras manos; por eso, al tratar de evaluar qué sistema numérico es realmente el más natural para usarlo, se descubre fácilmente que es el binario, porque con él se puede expresar normalmente cualquier concepto

que tenga dos estados opuestos, por ejemplo, Verdadero y Falso, Sí y No, Conectado y Desconectado, etc. Los sistemas de numeración (y en particular el binario) se estudian en el capítulo siguiente.

Ejemplo 1-7

Supóngase que se trabaja con un sistema de numeración de tres estados básicos o dígitos, el 0, el 1 y el 2. ¿Cómo expresaría en este sistema el número 5 decimal?

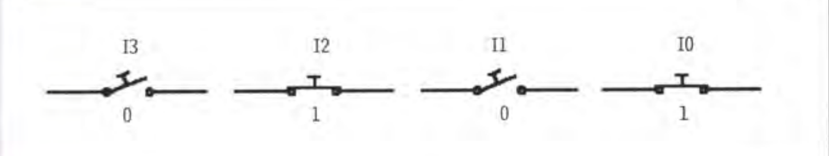
SOLUCIÓN

$5_{(10} = 12_{(3}$

1.7. Dispositivos binarios

Para poder representar una magnitud binaria se precisan dispositivos que admitan sólo dos estados de funcionamiento. El elemento binario más sencillo y claro es el interruptor de dos posiciones: abierto y cerrado (Off - On). Cuando el interruptor está abierto, representa el dígito binario 0 y cuando está cerrado, el 1. De esta forma, en la figura 1-6 se muestra la representación del número binario 0101 equivalente al 5 decimal, por medio de cuatro interruptores (I0, I1, I2 e I3).

Figura 1-6
Representación del número binario 0101 mediante interruptores.



Cualquier otro componente que admita solamente dos formas de funcionamiento puede servir para trabajar con Lógica Digital binaria. Así, un relé eléctrico (abierto/cerrado), una lámpara (encendida/apagada), un elemento magnetizable (magnetizado/desmagnetizado), etc., son dispositivos binarios.

Para la implementación de los elementos fundamentales de la lógica digital (puertas y memorias), se usa como dispositivo binario el ***transistor*** cuando trabaja en conmutación, es decir, con dos estados de funcionamiento: **corte** y **saturación**. Se elige este componente por su altísima velocidad en pasar de un estado a otro, lo que permite alcanzar muchos millones de conmutaciones por segundo (MHz).

Cuando el transistor trabaja en el punto de corte, no circula corriente por él y en su salida hay voltaje "alto", motivo por el cual se ha elegido este estado para representar el **dígito binario 1**. En saturación, al transistor le atraviesa la corriente máxima y en su salida el voltaje es nulo o "bajo", por lo que a esta situación se le asigna la representación del **dígito binario 0**. En la figura 1-7 se muestra la representación del número binario 0101 mediante cuatro transistores (T0, TI, T2 y T3).

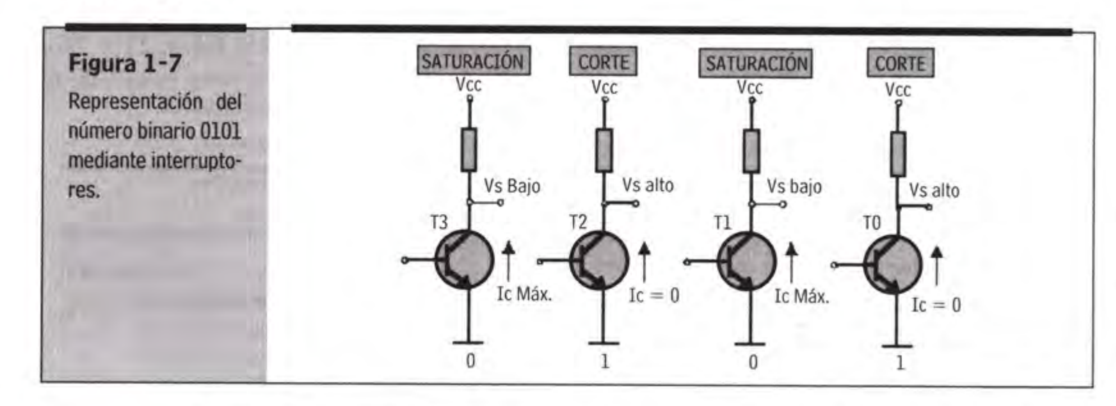


Figura 1-7
Representación del número binario 0101 mediante interruptores.

En realidad, los puntos de corte y saturación en los que trabajan los transistores en Lógica Digital no son tan rigurosos como se ha indicado hasta ahora. Esto significa que, cuando un transistor trabaja en corte, la corriente que le atraviesa no es nula sino muy pequeña, lo cual implica que la tensión de salida no sea exactamente V_{CC}, sino ligeramente menor debido al consumo de R_C. Figura 1-8.

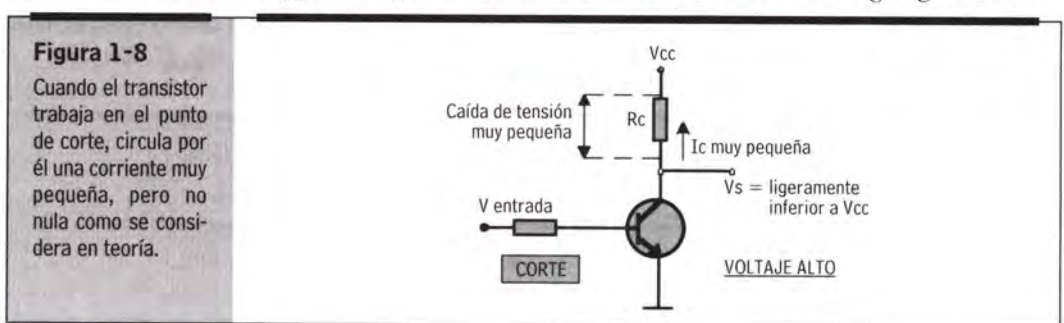


Figura 1-8
Cuando el transistor trabaja en el punto de corte, circula por él una corriente muy pequeña, pero no nula como se considera en teoría.

Teniendo en cuenta esta pequeña corriente en el punto de corte, el **voltaje alto** se considera con tensiones inferiores a V_{cc}.

Algo similar sucede en el punto de saturación en el que, teóricamente, se supone que pasa una corriente máxima que provoca en R_c una caída de tensión igual a V_{cc} con lo que la tensión de salida es nula. En la práctica, la corriente no alcanza ese valor máximo, sino algo menos, siendo la tensión de salida un poco mayor que 0 V. Figura 1-9.

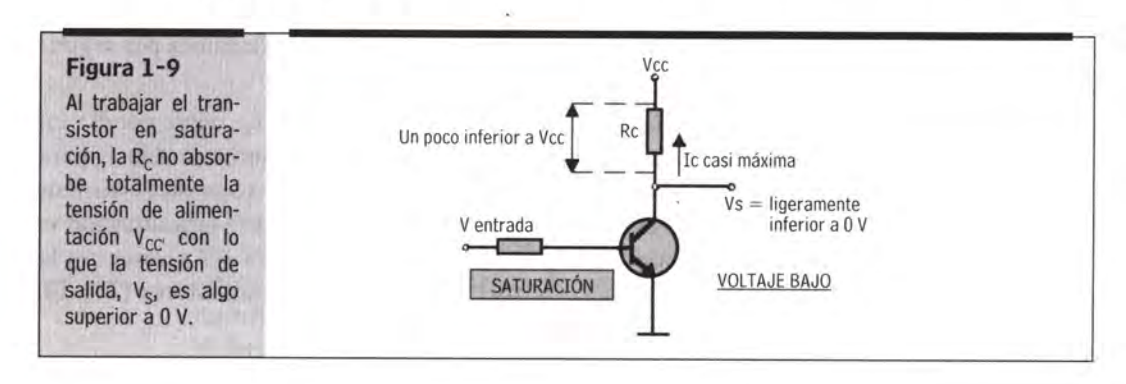


Figura 1-9
Al trabajar el transistor en saturación, la R_C no absorbe totalmente la tensión de alimentación V_{CC}, con lo que la tensión de salida, V_S, es algo superior a 0 V.

En los circuitos electrónicos digitales prácticos se trabaja con una $V_{CC} = +5$ V, generalmente, y se admite que hay "voltaje bajo" (que representa el dígito binario 0) cuando la tensión de salida del transistor no supera los 0,8 V. Por otra parte, el "voltaje alto" (que representa al dígito binario 1) puede tomar los valores comprendidos entre 2 y 5 V, de lo que se deduce que entre 0,8 V y 2 V no se debe trabajar en Electrónica Digital, al no tener asignado ningún estado válido dicho rango. Figura 1-10.

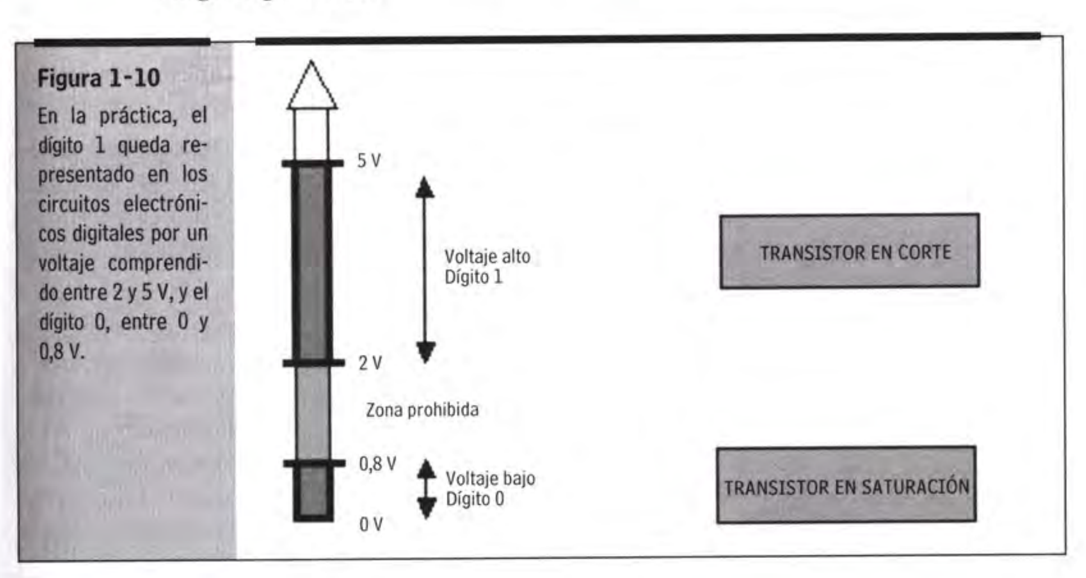


Figura 1-10

En la práctica, el dígito 1 queda representado en los circuitos electrónicos digitales por un voltaje comprendido entre 2 y 5 V, y el dígito 0, entre 0 y 0,8 V.

Ejemplo 1-8

Dado el circuito de la figura 1-11, indicar qué dígito binario representa el voltaje de salida del transistor.

(a) Cuando $I_C = 0,2$ mA

(b) Cuando $I_C = 0,4$ mA.

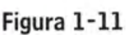
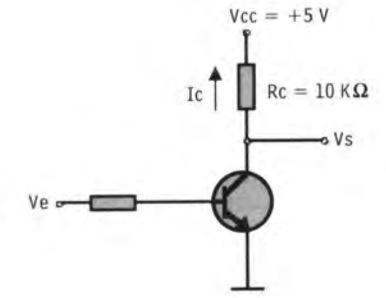


Figura 1-11

La tensión de salida V_S depende del valor de la corriente I_C.



SOLUCIÓN

(a) Dígito 1.

(b) Ninguno, puesto que $V_S = 1$ V y este valor está dentro de la zona prohibida.

1.8. Desarrollo de la lógica digital

Dentro del campo de la Electrónica, la Lógica Digital es relativamente moderna. Mientras las válvulas de vacío y los primitivos dispositivos electrónicos se remontan a principios de este siglo, los conceptos de la Lógica Digital no se desarrollaron como técnica independiente hasta finales de 1940, cuando se construyó el primer computador.

Inicialmente, los avances de la Electrónica Digital fueron muy lentos, porque sus elementos básicos tuvieron que ser construidos con válvulas de vacío que, dado el gran volumen, precio y complejidad de los circuitos anexos, resultaban prohibitivos en cuanto se requería una cierta cantidad de puertas.

La sustitución de la válvula de vacío por el transistor en 1950, redujo el tamaño de los circuitos considerablemente, pero el coste era aún alto. No obstante, entre 1950 y la primera mitad de 1960, la Lógica Digital se usó en computadores y en algunos circuitos electrónicos muy avanzados destinados al armamento militar.

El mayor impulso de la Electrónica Digital llegó con el descubrimiento del *circuito integrado* en 1960. Los circuitos integrados se adaptaron perfectamente a la Lógica Digital y proporcionaron los medios para la fabricación de bloques lógicos (equivalentes a un circuito compuesto por un número de válvulas de vacío de cuatro a ocho), con un tamaño inferior a un cuadrado de menos de 3 mm de lado. El aumento espectacular de la producción de circuitos integrados impulsó su reducción de precio y se produjo una explosión de la popularidad de la Lógica Digital entre los ingenieros de diseño electrónico y equipos de fabricación.

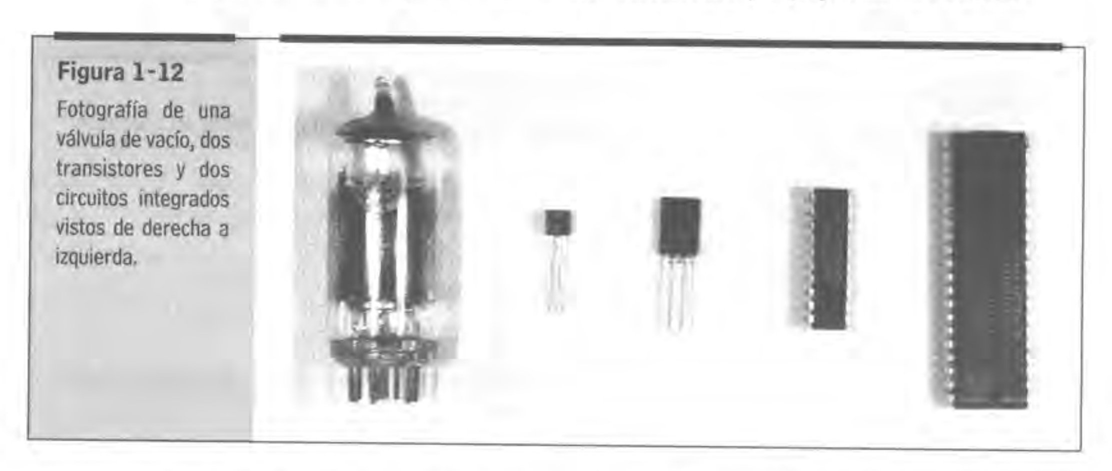


Figura 1-12
Fotografía de una válvula de vacío, dos transistores y dos circuitos integrados vistos de derecha a izquierda.

Todo lo descrito sucedió principalmente en la segunda mitad de la década de los 60, pudiendo decirse en el día de hoy que la Lógica Digital es la base de la Electrónica. Sin embargo, hubo que esperar a 1971 para poder construir en un circuito integrado la Unidad Central de Proceso de un computador; a dicho elemento se le llama *microprocesador* y es el componente fundamental en la fabricación de los modernos computadores.

Los efectos de la rápida evolución de los circuitos integrados digitales se han duplicado, en principio, porque los dispositivos digitales existentes disminuyeron drásticamente en tamaño y precio. En segundo lugar, al extenderse el empleo de la Lógica Digital, se comenzó a utilizar en cosas completamente nuevas, produciendo nuevos productos y reemplazando a otros que, hasta ese momento, utilizaban circuitos analógicos, como los receptores de radio y televisores domésticos, que no se pueden transformar en circuitos digitales sin reestructurar los modelos de las industrias que desarrollaron esos productos. Sin embargo, en los últimos años, el número de este tipo de componentes analógicos ha decrecido extraordinariamente.

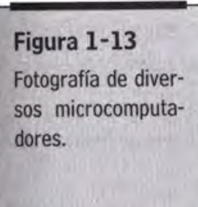


Figura 1-13
Fotografía de diversos microcomputadores.



También se puede analizar la Lógica Digital dividiéndola en tres categorías de productos a los que se aplica. En primer lugar, se sitúan los **computadores**, que, aunque en un principio los hubo también de tecnología analógica, en la actualidad sólo se fabrican digitales; en segundo lugar, existen multitud de **dispositivos periféricos**, que reciben y proporcionan la información de los computadores. Y, por último, hay una amplísima gama de **productos diversos** como pueden ser los aparatos de medida, de tipo doméstico, de tipo industrial, etc. No se deben menospreciar las dos últimas categorías porque los computadores son cada vez más pequeños y potentes, hasta el punto de poderse encontrar clasificados o formando parte de los elementos de las dos últimas categorías, es decir, dentro de los aparatos de medida y de los dispositivos periféricos. Por otro lado, aparatos de uso común, como la calculadora de bolsillo, están llegando incluso a actuar como verdaderos computadores. Resumiendo, los computadores han contribui-

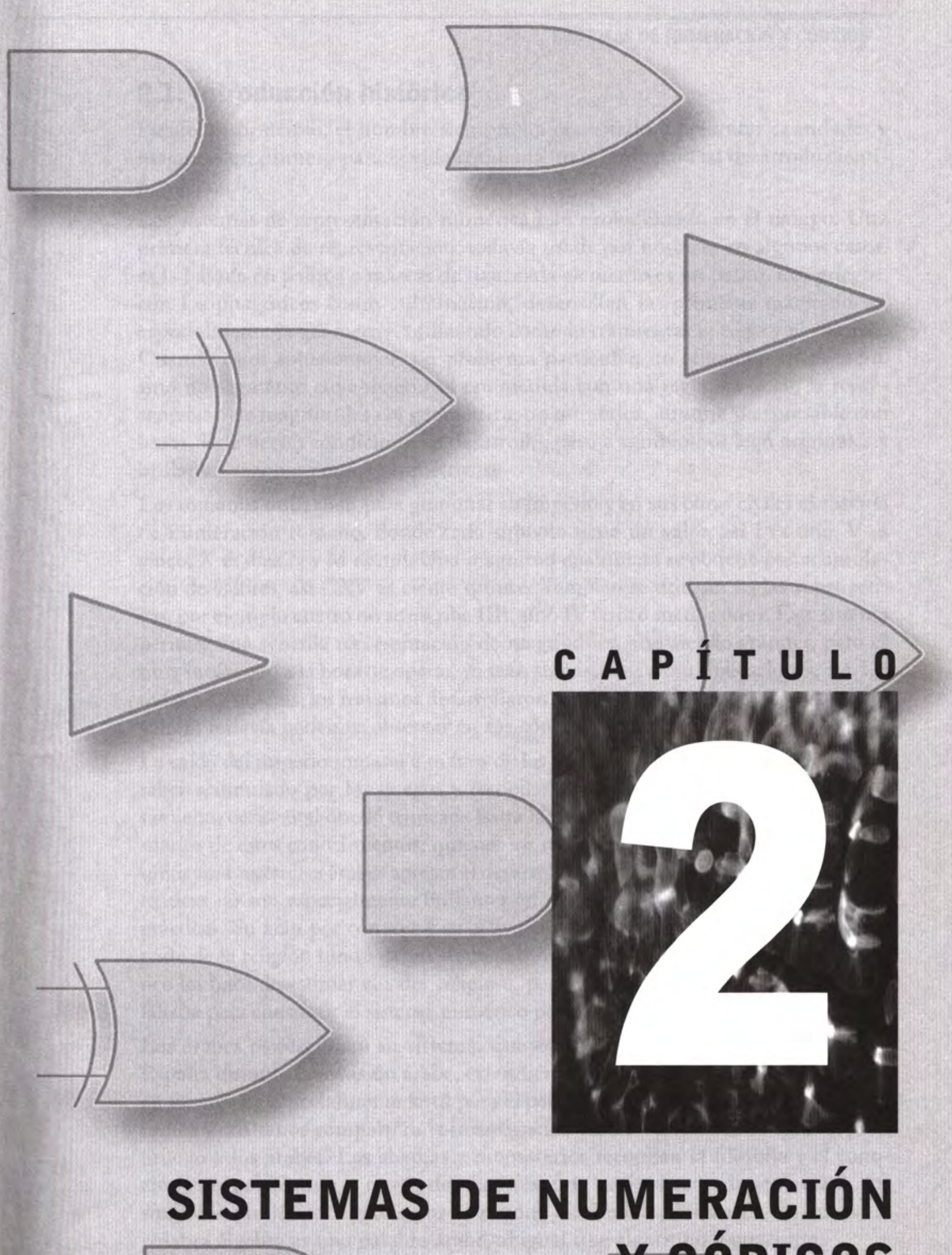
do muy significativamente al desarrollo de la Lógica Digital electrónica, si bien, en estos momentos, tales técnicas son cada vez más aplicables a todo tipo de productos.

Aun con el nivel de sofistificación que la Lógica Digital ha alcanzado, "cambio" es su palabra clave. En Lógica Digital, como en otros campos en los que el hombre investiga, hay una secuencia cíclica de acontecimientos. Primero, hay una etapa de creación; después, un estado de maduración y, por último, un declive. La Lógica Digital está situada en el centro de una fase muy activa de su estado de crecimiento durante el cual son normales los cambios constantes y las innovaciones. Los circuitos integrados digitales, en especial los comprendidos en el campo de los elementos de memoria, se desarrollan tan rápidamente que a menudo resultan anticuados antes de ser fabricados y comercializados. Por todo lo expuesto, en esta obra se trata de fotografiar el instante actual del proceso evolutivo de la Lógica Digital.

Los circuitos integrados tienen una gran expectativa de futuro y su desarrollo se orienta especialmente en dos direcciones claves:

1ª) Una tendencia hacia la **reducción de tamaño** hasta conseguir que los sistemas más complejos puedan reemplazarse por un simple circuito integrado. Esto supone la integración de millones de transistores en un chip.

2ª) Un **aumento en la potencia de cómputo**, en particular referido al incremento de la capacidad de memoria de los componentes destinados a tal fin, y de la velocidad de funcionamiento, que ya supera los miles de millones de ciclos por segundo.



CAPÍTULO 2

SISTEMAS DE NUMERACIÓN Y CÓDIGOS

2.1. Introducción histórica

Desde un principio, el hombre siempre ha necesitado representar cantidades y magnitudes, primero para la vida cotidiana, más tarde para su desarrollo científico-técnico.

Los sistemas de representación numérica han evolucionado en el tiempo. Una primera técnica de representación, todavía usada por nosotros en algunos casos, es la basada en palitos o marcas de tiza; cada elemento es un palito. Los griegos, con los pitagóricos como culminación, desarrollan las primeras matemáticas, especialmente la geometría, utilizando como herramientas la regla y el compás. Cuando éstos solucionaban un problema particular, no obtenían un número, sino un segmento cuya magnitud era medida con una regla. Es decir, la representación de magnitudes era geométrica, no numérica, aunque sí expresable con letras. Este hecho condicionó su desarrollo, pero a cambio nos legó originales y brillantes razonamientos geométricos.

Los romanos utilizaban para gestionar su imperio y en sus obras civiles el sistema de numeración romano, donde cada símbolo tiene un valor, así I es uno, V es cinco, X es diez..., y M es mil. Una magnitud cualquiera se obtiene por acumulación de valores, así CXV es ciento quince. También se utilizan reglas substractivas, por ejemplo cuatro no se escribe IIII, sino IV (cinco menos uno). Este sistema permite una sencilla representación de magnitudes, aun siendo grandes, pero es muy incómodo a la hora de operar, ya sean sumas, restas, multiplicaciones, etc. De todas las maneras, los romanos desarrollaron técnicas para operar cuyos resultados válidos todavía podemos observar en sus construcciones civiles.

La caída del imperio romano a manos de los bárbaros conllevó el olvido de todo el saber acumulado por los griegos y recopilado por los romanos; la línea de pensamiento occidental quedó truncada hasta la Edad Media. Son los árabes los depositarios de estos conocimientos, quienes los recopilan, ordenan y en algunos casos unifican. Los emires árabes apoyan el desarrollo de las ciencias y las artes; sus matemáticas no son especialmente brillantes en nuevos avances, sino eminentemente prácticas. Su afán por conocer y su activo comercio les lleva a admirar el uso por parte de la religión hindú de un símbolo que representa la nada. Su espíritu práctico les hace desestimar el valor religioso, para reconocer que ésta es la pieza que faltaba para constituir el sistema numérico posicional de base diez: el cero.

Los árabes perfeccionan su sistema, que es introducido en Europa a través de España durante la invasión árabe, extendiéndose y consolidándose rápidamente en una época especialmente fértil para el pensamiento español, en la que árabes, judíos y cristianos compartían la investigación técnico-científica. Debemos pues mucho a los árabes. Las abadías y monasterios recopilan la filosofía y el conocimiento grecolatino a través de ellos (en una curiosa retroalimentación), los símbolos numéricos usados por el mundo occidental derivan de los suyos, la palabra álgebra es una palabra árabe, al igual que algoritmo o guarismo.

Si bien sabemos que es a los árabes a los que cabe asignar la gloria y asentamiento del sistema decimal, todavía queda una cuestión por resolver: ¿por qué base 10? La respuesta no parece ser otra que la existencia de 10 dedos. Ésta no ha sido la

única base utilizada, por ejemplo otro pueblo clave en la historia de las matemáticas, los babilonios, utilizaban la base 60, mejor que la base 10 para realizar las particiones tan necesarias en el comercio, ya que 60 es divisible por 2, 3, 4, 5, 6, 10, 12, 15, etc. Esta base se ha perpetuado en nuestros días en algunos casos como la medición del tiempo, o de los ángulos. También la base 12 ha sido muy utilizada, recordemos solamente la palabra docena y sus usos. En cuanto al sistema binario, éste ha sido utilizado por tribus primitivas de América del Sur y Oceanía.

El sistema decimal es utilizado sin discusión en todas las facetas de la vida diaria, sin modificarse; la aparición de los sistemas digitales conlleva el uso del sistema binario -planteado por primera vez por Leibniz en el siglo XVII- y el desarrollo de técnicas de representación basadas en éste.

2.2. Sistemas de numeración

El sistema de numeración decimal consta de 10 símbolos (dígitos), uno para cada uno de los nueve números 1 ... 9 y otro para el cero. Además es necesaria una regla de uso de los símbolos para representar magnitudes que en este caso es posicional: a cada símbolo le corresponde un valor según la posición que ocupe. Recordemos que el sistema romano no es posicional, M siempre vale mil cualquiera que sea su posición.

Antes de continuar describiendo los sistemas decimal y binario planteemos la pregunta: ¿cuántos símbolos son necesarios?

Por ejemplo, para leer un periódico es necesario dominar entre 3.000 y 4.000 palabras, siendo 8.000 las necesarias para un texto científico; así los chinos y japoneses han de conocer ese número de signos distintos (habilidad que les condiciona en su formación), mientras que los europeos deben conocer combinaciones de un menor número de símbolos, los 26 del alfabeto romano.

Si trasladamos los 8.000 signos al campo de los dígitos, bastarían palabras de cuatro dígitos decimales para representarlos, mientras que en binario serían necesarias palabras de trece dígitos. Las reglas de formación de palabras serían mucho más sencillas en binario que en decimal, mientras la lectura sería más sencilla en decimal que en binario. Por otra parte, para sumar números decimales hay que saber una tabla con 100 casos que se reducen a cuatro si los números son binarios.

La discusión se plantea en cuanto a: economía de símbolos (capacidad de almacenamiento), reglas de uso, rapidez de uso y facilidad operativa.

En la vida diaria utilizamos el decimal sin discusión, pero ¿por qué se usa el binario en los sistemas digitales? Las razones son varias:

- Física: actualmente los dispositivos eléctricos y mecánicos sólo presentan con facilidad dos estados distintos entre sí. Interruptor: cerrado-abierto, transistor: corte-saturación, etc.
- Lógica: las reglas de la lógica clásica son de tipo binario: verdadero-falso.
- Operatividad: las reglas de cálculo binario son muy sencillas. Los sistemas digitales *prefieren* muchas operaciones sencillas que pocas complejas, al contrario que el hombre.

 © ITES - PARANINFO

- Representación: los códigos binarios tienen propiedades especiales.

Como contrapartida, los sistemas digitales deben manejar una gran cantidad de dígitos binarios y deben hacerlo de una forma rápida, pero si algo distingue histórica y actualmente a los sistemas digitales es su gran capacidad de almacenamiento y su velocidad de proceso.

2.2.1. Conversión entre sistemas numéricos

En los anteriores párrafos hemos descrito someramente los distintos sistemas de numeración; se hace necesaria una descripción más formal, amén de las reglas que permitan la conversión de números entre distintos sistemas numéricos.

Inicialmente, todos los sistemas numéricos son posicionales (otra cosa serán los códigos), correspondiéndole a cada dígito un peso según su posición. La distribución de estos pesos es arbitraria, aunque parece lógico que sea creciente según un criterio constante, por ejemplo la progresión geométrica.

Así, en el sistema decimal la primera posición (de derecha a izquierda) tiene un peso de 10^0, la siguiente de 10^1, y así sucesivamente según la razón de 10; del mismo modo los pesos binarios son 2^0, 2^1, 2^2, etc. Este tipo de distribución parece evidente y única, pero pensemos en el tiempo, en los ángulos, etc.

Ejemplo 2-1

Convertir 2 días, 14 horas y 34 minutos (2:14:34) en minutos.

2:14:34 = 2 x (24 x 60) + 14 x (60) + 34 x (1) = 3.754 minutos.
En este caso los pesos son 1, 60 y 24 x 60.

2.2.2. Notación radical, poliádica o polinómica

Teniendo en cuenta lo anterior, un sistema de base B (B>1) es de notación radical, poliádica o polinómica si sus pesos crecen de forma geométrica. Un número N de base B con n dígitos se expresa de la siguiente forma.

$$N_{CB} = \sum_{i=0}^{n-1} D_i \cdot B^i = D_{n-1} B^{n-1} + D_{n-2} B^{n-2} + \cdots + D_1 B^1 + D_0 B^0 \qquad (2.1)$$

Ejemplo 2-2

Descomponer los siguientes números:

$$258_{C10} = \sum_{i=0}^{2} D_i \bullet B^i = 2 \times 10^2 + 5 \times 10^1 + 8 \times 10^0$$

$$10110_{C2} = \sum_{i=0}^{4} D_i \bullet B^i = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

Si el número N es real, con n dígitos enteros y m dígitos decimales, su notación poliádica es:

$$N_{CB} = \sum_{i=-m}^{n-1} D_i \cdot B^i = D_{n-1} B^{n-1} + \cdots + D_1 B^1 + D_0 B^0 + D_{-1} B^{-1} + D_{-2} B^{-2} + \cdots + D_{-m} B^{-m} \quad (2.2)$$

Ejemplo 2-3

Descomponer el siguiente número con parte fraccionaria.

$$258{,}37_{C10} = \sum_{i=0}^{2} D_i \bullet B^i = 2 \times 10^2 + 5 \times 10^1 + 8 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2}$$

Ambos lados de la igualdad son idénticos: la parte derecha es una representación explícita de la magnitud, mientras que la izquierda es una notación compacta con las potencias de 10 implícitas.

2.2.2.1. Paso de binario a decimal

Un número N_2 en binario se convierte en N_{10} según el sumatorio:

$$N_2 = \sum_{i=-m}^{n-1} D_i \cdot B^i = N_{10}, \text{ donde } B = 2 \quad (2.3)$$

Ejemplo 2-4

Representar en decimal los siguientes números binarios.

$$1011001_2 = \sum_{i=0}^{6} D_i \bullet B^i = 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 89_{10}$$

$$101{,}001_2 = \sum_{i=-3}^{2} D_i \bullet B^i = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 5{,}375_{10}$$

Generalizando, para pasar de cualquier base a decimal:

$$N_{CB} = \sum_{i=-m}^{n-1} D_i \cdot B^i = N_{10},$$

donde B es el valor de la base (2.4)

Ejemplo 2-5

Representar en decimal los siguientes números.

$$321_5 = \sum_{i=0}^{2} D_i \cdot B^i = 3 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 = 86_{10}$$

$$61{,}72_8 = 6 \times 8^1 + 1 \times 8^0 + 7 \times 8^{-1} + 2 \times 8^{-2} = 49{,}906_{10}$$

Si el número binario no tiene muchos dígitos, la transformación se hace *con la vista*, según la tabla 2-1.

Tabla 2-1

Potencias de 2 para la conversión binario-decimal, y viceversa.

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
1.024	512	256	128	64	32	16	8	4	2	1	0,5	0,25	0,125

2.2.2.2. Paso de decimal a binario

Si el número decimal es entero, se utiliza la división sucesiva:

1. Se divide el número original entre 2.
2. El cociente resultante se divide sucesivamente entre 2, hasta que éste sea menor que 2.
3. El número binario se obtiene leyendo inversamente el último cociente y todos los restos.

Ejemplo 2-6

Escribir 23_{10} en base 2.

23:2 = 11 Resto = **1**
11:2 = 5 Resto = **1**
5:2 = 2 Resto = **1**
2:2 = **1** Resto = **0**
$23_{10} = 10111_2$

Esta técnica es sencilla pero lenta; en general, si el número decimal es pequeño su correspondiente binario se obtiene utilizando la tabla 2-1.

Si el número decimal es real, la parte fraccionaria se obtiene mediante la multiplicación sucesiva:

1. Se toma la parte fraccionaria decimal y se multiplica por 2.
2. La parte fraccionaria (sin el entero) del anterior resultado es multiplicada por 2. Este proceso se repite hasta que la parte fraccionaria obtenida sea nula, o hasta obtener la precisión deseada.
3. La parte fraccionaria binaria se obtiene leyendo los bits de arriba abajo.

Ejemplo 2-7

Escribir $23{,}57_{10}$ en base dos.

Parte entera $23_{10} = 10111_2$.

Parte fraccionaria $0{,}57_{10} = 0{,}10010_2$.
0,57 x 2 = **1**,14
0,14 x 2 = **0**,28
0,28 x 2 = **0**,56x
0,56 x 2 = **1**,12
0,12 X 2 = **0**,24

Conjuntando ambos resultados $23{,}57_{10}=10111{,}10010_2$.

Por supuesto, el número de dígitos fraccionarios no tiene por qué coincidir, pudiendo ser infinitos los dígitos binarios necesarios.

Generalizando, para pasar de decimal a cualquier base hay que sustituir en el anterior método el 2 por la base apropiada.

Ejemplo 2-8

Escribir $123{,}57_{10}$ en base 6.

Parte entera:
123:6 = 20 Resto = **3**
20:6 = **3** Resto = **2**
$23_{10} = 323_6$

Parte fraccionaria
0,57x6 = **3**,42
0,42x6 = **2**,52
0,52x6 = **3**,12
$0{,}57_{10} = 0{,}323_6$

Conjuntando ambos resultados: $123{,}57_{10} = 323{,}323_6$

Para pasar de una representación en base B1 a otra en base B2, lo más cómodo es pasar de la primera a decimal y de ésta a la correspondiente en base B2, según las técnicas expuestas.

2.2.3. Sistema binario

El sistema binario tiene como símbolos el 0 y el 1 y para representar magnitudes utiliza la notación poliádica de base 2. Un bit es la cantidad de información contenida en un dígito binario, el término se obtiene como el acrónimo de las palabras inglesas binary digit. Así, decimos que 101 tiene tres bits.

En la tabla 2-2 vemos representados en binario los trece primeros números decimales:

Tabla 2-2
Sistema binario.

Decimal	Binario	Binario 4 bits
0	0	0000
1	1	0001
2	10	0010
3	11	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111
8	1000	1000
9	1001	1001
10	1010	1010
11	1011	1011
12	1100	1100

Esta tabla nos permite plantear los conceptos de rango y longitud fija y variable. El número de dígitos decimales utilizados depende de la magnitud a representar y procesar; el cerebro procesa sin problemas números de longitud variable, es más, nos extrañaría ver 00235. Sin embargo, los sistemas digitales se comportan al contrario, prefieren manejar siempre agrupaciones de un número fijo de bits aunque algunos de ellos no aporten información, por ejemplo el binario de cuatro bits representa 2 como 0010.

Lo anterior supone que hay que elegir un número de bits, que se mantendrá fijo cualquiera que sea el número a representar, por lo que se impone una cota superior de representación que con longitud variable no existía. Por ejemplo, con 4 bits se representa de 0000 (0) a 1111 (15), la cota superior es 15. Si se utilizan m bits se obtienen 2^m combinaciones que se reflejan en ese rango de representación que va de 0 a 2^m-1.

Rango con m bits = [0 a 2^m-1] (2.5)

Cuando diseñamos un sistema digital, es definitivo conocer el rango de valores que se va a procesar, ya que nos indicará el número de bits a utilizar en la representación binaria.

En computadores -los sistemas digitales más comunes- el número de bits es fijo, lo que conlleva un rango, que por ser suficientemente grande no suele dar problemas, pero en algunos casos aparece el mensaje "*error overflow*", síntoma inequívoco de que se ha desbordado la capacidad de cálculo del computador.

2.2.4. Sistemas octal y hexadecimal

Estos sistemas de representación son utilizados para presentar información binaria de forma compacta (por ejemplo en volcados de memoria), ya que la vista *prefiere ver* pocos símbolos, aunque variados, que muchos símbolos binarios.

Ejemplo 2-9

Escribir el número 101001001011 en octal y hexadecimal.

$101001001011 = 5113_{C8} = A4B_{C16}$

La transferencia entre octal y hexadecimal a binario es sencilla: tres bits se convierten en un dígito octal y cuatro bits se convierten en un dígito hexadecimal, y viceversa. Si faltaran o sobraran dígitos, se considerarán ceros.

La tabla 2-3 relaciona los sistemas decimal, binario de cuatro bits, octal y hexadecimal. El sistema hexadecimal consta de 16 símbolos, por tanto hay que añadir seis símbolos: A, B, C, D, E y F a los diez decimales.

Tabla 2-3
Sistemas binario, octal y hexadecimal.

Dec.	Binario	Bin. 4 bits	Octal	Hex.
0	0	0000	0	0
1	1	0001	1	1
2	10	0010	2	2
3	11	0011	3	3
4	100	0100	4	4
5	101	0101	5	5
6	110	0110	6	6
7	111	0111	7	7
8	1000	1000	10	8
9	1001	1001	11	9
10	1010	1010	12	A
11	1011	1011	13	B
12	1100	1100	14	C
13	1101	1101	15	D
14	1110	1110	16	E
15	1111	1111	17	F

Para pasar de binario a octal o hexadecimal sólo hay que agrupar bits, teniendo que desagruparlos para pasar de octal o hexadecimal a binario.

Ejemplo 2-10

Convertir números de una base a otra.

Octal:			
001	100	101	111
1	4	5	7

Hexadecimal:		
0011	0010	1111
3	2	F

$001100101111_2 = 1457_8 = 32F_{16}$

Octal:		
111	100	110
7	4	6

Hexadecimal:		
0001	1110	0110
1	E	6

$111100110_2 = 746_8 = 1E6_{16}$

Hexadecimal:			
A	3	8	B
1010	0011	1000	1011

$A38B_{16} = 1010001110001011_2$

Octal:		
3	4	7
011	100	111

$347_8 = 011100111_2$

El paso entre octal y hexadecimal, y viceversa, no es muy común y se hará utilizando el binario o el código decimal como auxiliar.

2.3. Representación de números con signo

Los números, además de magnitud, pueden tener signo, para ello utilizamos un nuevo símbolo, el signo: + ó -. Sin embargo, el sistema binario tiene dos símbolos, 0 y 1, y no tiene capacidad para incluir el signo, es decir, un número con signo tendrá que ser representado con ceros y unos. Existen varias formas básicas para esta representación con signo:

- Representación con signo y magnitud.
- Representación con complemento a 1.
- Representación con complemento a 2.
- Representación mediante exceso o sesgo.

Si existen distintas representaciones, quiere decir que esa misma secuencia de bits puede ser interpretada de forma distinta -esta ambigüedad nunca se da en decimales-, debiendo indicarse siempre cuál es el criterio utilizado. Por ejemplo, la secuencia 1001, es 9 en binario puro sin signo, es -1 en binario con signo y magnitud, es -6 en binario con complemento a 1, es -7 en binario con complemento a 2 y es +1 en binario con exceso a 8.

Queda describir cada modo de representar magnitudes con signo.

2.3.1. Criterio de signo y magnitud

Este criterio responde a un razonamiento intuitivo: se añade arbitrariamente a la magnitud un bit de signo. Un 0 como signo positivo y un 1 como signo negativo. El número binario queda conformado por:

- 1 bit de signo.
- n bits de magnitud.

Ejemplo 2-11

Interpreta los números en binario con bit de signo.

1001 = -1
0110 = +6
011 = +3
101011 = -11

La tabla 2-4 interpreta el código binario con bit de signo y magnitud, un bit de signo y tres de magnitud.

Tabla 2-4

Código binario magnitud y signo.

Decimal	Signo y magnitud
+0	0 000
+1	0 001
+2	0 010
+3	0 011
+4	0 100
+5	0 101
+6	0 110
+7	0 111
-0	1 000
-1	1 001
-2	1 010
-3	1 011
-4	1 100
-5	1 101
-6	1 110
-7	1 111

Anotemos las características principales de este código:

- Es fácil de interpretar.
- Negar un número supone invertir el bit de signo.
- Con n bits el rango es: $-(2^{n-1}-1)$ a $+(2^{n-1}-1)$. Con 4 bits de -7 a +7.
- Existen el +0 y el -0: 0000 y 1000, respectivamente.
- Es incómodo para operar.
- El cero no está centrado respecto a los números positivos y negativos.

2.3.2. Criterio de los complementos

Un número negativo puede ser escrito como el complemento del correspondiente positivo, donde el complemento es una operación como la suma o la resta.

Se puede obtener el complemento de un número escrito en cualquier base, y éste puede ser de dos tipos:

Complemento a la base:

- Se forma el número base con un 1 seguido de tantos ceros como dígitos tenga el número a complementar.
- El complemento se obtiene restando el número base menos el número a complementar.
- El complemento tiene tantos dígitos como el original.

Complemento a la base -1:

- Se forma el número base con tantos dígitos de valor la base menos 1 como tenga el número a complementar.
- El complemento se obtiene restando el número base menos el número a complementar.
- El complemento tiene tantos dígitos como el original.

Ejemplo 2-12

Obtener los complementos a 9 y a 10 de 13, 104 y 0 y los complementos a 2 y a 1 de 011, 0100 y 00.

Complemento a 10:	
100 - 13 = 87	C-10(13) = 87
1000 - 104 = 896	C-10(104) = 896
10 - 0 = 10	C-10(0) = 10
Complemento a 9:	
99 - 13 = 86	C-9(13) = 86
999 - 104 = 895	C-9(104) = 895
9 - 0 = 9	C-9(0) = 9
Complemento a 2:	
1000 - 011 = 101	C-2(011) = 101
10000 - 0100 = 1100	C-2(0100) = 1100
100 - 00 = 100	C-2(00) = 00
Complemento a 1	
111 - 011 = 100	C-1(011) = 100
11111 - 0100 = 1011	C-1(0100) = 1011
11 - 00 = 11	C-1(00) = 11

En el caso del sistema binario las reglas de obtención del complemento son más sencillas:

- Complemento a 1: se obtiene invirtiendo los ceros por unos y viceversa del número a complementar, es decir, negando el número original.
- Complemento a 2: se obtiene sumando 1 al complemento a 1 del número a complementar. El C-2 y el C-1 se distinguen en 1. Otra técnica (más sencilla que la anterior) se basa en la reescritura: el número a complementar a 2 se reescribe de derecha a izquierda copiando los bits hasta el primer uno, éste incluido, y a partir de él los siguientes bits son negados.

Ejemplo 2-13

Obtener el complemento a 1 y a 2 de 0100, 1100 y 01.

Complemento a 1:
C-1(0100) = 1011
C-1(1100) = 0011
C-1(01) = 10
Complemento a 2:
C-2(0100) = C-1(0100) + 1 = 1011 + 1 = 1100
C-2(1100) = C-1(1100) + 1 = 0011 + 1 = 0100
C-2(01) = C-1(01) + 1 = 10 + 1 = 11

Utilizando la técnica del complemento, los números con signo se escriben como sigue:

- Los números positivos, ya sea utilizando el C-1 o C-2, se escriben como la magnitud en binario puro precedida por un cero (signo +).
- Los números negativos se escriben como el complemento a 1 o a 2 del correspondiente positivo, es decir, en este código complementar es invertir.

Lo anterior supone que un número positivo se escribe igual en binario puro con C-1 que en binario puro con C-2 (y a su vez igual que en binario puro con bit de signo), mientras que un número negativo se escribe distinto con C-1 que con C-2. Además es reseñable que cualquier número binario así codificado tiene al menos dos bits.

Ejemplo 2-14

Codifica en binario con C-1 y C-2 los distintos números.

	+7	+39	+1	+0
Binario C-1	0111	0100111	01	00
Binario C-2	0111	0100111	01	00

	-7	-39	-1	-0
Binario C-1	1000	1011000	10	11
Binario C-2	1001	1011001	11	00

En la tabla 2-5 observamos qué interpretación decimal corresponde a cada combinación binaria, si ésta es leída en binario puro, en binario con C-1 o en binario con C-2.

Tabla 2-5

Binario con y sin signo.

	Leído como Binario		
Binario	sin signo	con C-1	con C-2
0000	0	+0	+0
0001	1	+1	+1
0010	2	+2	+2
0011	3	+3	+3
0100	4	+4	+4
0101	5	+5	+5
0110	6	+6	+6
0111	7	+7	+7
1000	8	-7	-8
1001	9	-6	-7
1010	10	-5	-6
1011	11	-4	-5
1100	12	-3	-4
1101	13	-2	-3
1110	14	-1	-2
1111	15	-0	-1

Vemos que aunque se parecen, existen diferencias entre el C-1 y el C-2. Las principales características de la codificación de números con signo en binario puro con C-1 son:

- Su interpretación decimal no es inmediata.
- Negar un número supone obtener su complemento a 1.
- Con n bits, el rango es: $-(2^{n-1}-1)$ a $+(2^{n-1}-1)$. Para 8 bits: -127 a +127.
- Existen el +0 y el -0: 0000 y 1111, respectivamente.
- Es relativamente cómodo para operar.
- El cero no está centrado respecto de los números positivos y negativos.

Las principales características de la codificación de números con signo en binario puro con C-2 son:

- Su interpretación decimal no es inmediata.
- Negar un número supone obtener su complemento a 2.
- Con n bits, su rango es: -2^{n-1} a $+(2^{n-1}-1)$. Por ejemplo, para 8 bits: -128 a +127.
- Sólo existe el 0: 0000.

- Es cómodo para operar.
- El cero no está centrado respecto de los números positivos y negativos.

Comparando el C-1 y el C-2, las diferencias se dan, aparte del propio complemento, en el rango y en la comodidad para operar.

2.3.3. Criterio de exceso o con sesgo

En este caso el código se forma sumando cierto valor al número original, de ahí su nombre. Para codificar un número decimal con signo en binario con sesgo se procede como sigue:

- Al número decimal se le suma cierta cantidad -sesgo, 'bias' o exceso-. Este valor suele ser 2^{n-1} o 2^{n-1}-1, donde n es el número de bits.
- Este nuevo número obtenido es codificado en binario puro, convirtiéndose en el número codificado.

En el caso inverso, para obtener el número decimal con signo correspondiente al binario con sesgo se procede como sigue:

- El número binario es interpretado en decimal como si se tratara de binario puro.
- Al número decimal obtenido se le resta el sesgo del código, el número resultante es el correspondiente decimal.

Ejemplo 2-15

Transforma los números +3 y -5 a binario de cuatro bits con sesgo.

Realiza la operación inversa para 1111 y 0101.

Binario con exceso a 7
XS7(+3) = BP(+3+7) = 1010
XS7(-5) = BP(-5+7) = 0010
1111 = 15 - 7 = 8
0101 = 5 - 7 = -2

Binario con exceso a 8
XS8(+3) = BP(+3+8) = 1011
XS8(-5) = BP(-5+8) = 0011
1111 = 15 - 8 = 7
0101 = 5 - 8 = -3

En la tabla 2-6 se muestra la codificación en binario puro con sesgo y complemento para los números decimales del -8 al +8.

Las principales características de la codificación de números con signo en binario con sesgo son:

- Su interpretación decimal no es inmediata.
- Negar un número no es directo, supone obtener el C-1 del número anterior o posterior de la tabla según sea XS-7 o XS-8, respectivamente.

- Con n bits y exceso 2^{n-1}-1, el rango es: $-2^{n-1} + 1$ a $+ 2^{n-1}$, así para 8 bits con exceso a 127 es: -127 a +128
- Con n bits y exceso 2^{n-1} el rango es: -2^{n-1} a $+(2^{n-1}-1)$, así para 8 bits con exceso a 128 es: -128 a +127.
- Sólo existe el 0, ya sea con un sesgo u otro.
- El cero está centrado respecto de los valores positivos y negativos.
- Se utiliza para representar el exponente de la representación en coma flotante.

Tabla 2-6

Codificación de decimales con signo según distintos códigos.

Decimal	XS-7	XS-8	B C-1	B C-2	M-S †
+8	1111				
+7	1110	1111	0111	0111	0111
+6	1101	1110	0110	0110	0110
+5	1100	1101	0101	0101	0101
+4	1011	1100	0100	0100	0100
+3	1010	1011	0011	0011	0011
+2	1001	1010	0010	0010	0010
+1	1000	1001	0001	0001	0001
+0	0111	1000	0000	0000	0000
-0			1111		1000
-1	0110	0111	1110	1111	1001
-2	0101	0110	1101	1110	1010
-3	0100	0101	1100	1101	1011
-4	0011	0100	1011	1100	1100
-5	0010	0011	1010	1011	1101
-6	0001	0010	1001	1010	1110
-7	0000	0001	1000	1001	1111
-8		0000		1000	

† Magnitud y Signo

En la tabla 2-7 se indica qué valor decimal corresponde a cada combinación binaria, según sea interpretado bajo un criterio u otro.

La tabla 2-8 a modo de resumen muestra las coincidencias y diferencias entre los distintos códigos binarios con signo presentados.

Tabla 2-7
Interpretación de distintas secuencias binarias según distintos códigos.

Binario	M-S	C-1	C-2	XS-7	XS-8
0000	+0	+0	+0	-7	-8
0001	+1	+1	+1	-6	-7
0010	+2	+2	+2	-5	-6
0011	+3	+3	+3	-4	-5
0100	+4	+4	+4	-3	-4
0101	+5	+5	+5	-2	-3
0110	+6	+6	+6	-1	-2
0111	+7	+7	+7	0	-1
1000	-0	-7	-8	+1	0
1001	-1	-6	-7	+2	+1
1010	-2	-5	-6	+3	+2
1011	-3	-4	-5	+4	+3
1100	-4	-3	-4	+5	+4
1101	-5	-2	-3	+6	+5
1110	-6	-1	-2	+7	+6
1111	-7	-0	-1	+8	+7

Tabla 2-8
Comparación entre distintos códigos binarios con signo.

	Binario C-1	Binario C-2	Binario sesgo	Mag. Signo
Interpretación	No inmediata	No inmediata	No inmediata	Inmediata
Negación	C-1. Sencillo	C-2. Sencillo	No sencillo	Bit signo. Fácil
Rango	$-2^{n-1}+1$, $+2^{n-1}-1$	-2^{n-1} , $+2^{n-1}-1$	-2^{n-1} , $+2^{n-1}-1$ ó $-2^{n-1}+1$, $+2^{n-1}$	$-2^{n-1}+1$, $+2^{n-1}-1$
Duplicidad 0	SÍ	NO	NO	SÍ
Simetría 0	NO	NO	SÍ	NO
Operatividad	Sencilla	Sencilla	Normal	Complicada

2.4. Operaciones aritméticas básicas

La suma y la resta son las operaciones básicas de la aritmética binaria, y en ellas se basan otras muchas. Las operaciones suma y resta se definen mediante las tablas de la suma y de la resta binaria.

BIBLIOTECA CADIZ PUBLICA

Las reglas de la suma y resta de la tabla 2-9 son válidas para operaciones con números codificados en binario puro, por tanto sin posibilidad de signo. Para sumar números con signo es necesario utilizar los códigos ya descritos anteriormente -binario con C-1, C-2, sesgo y bit de signo-, y encontrar la forma de sumarlos de manera que el resultado sea correcto, teniendo en cuenta que la regla de la suma es válida para binario puro, pero no tiene por qué serlo para los otros códigos.

Tabla 2-9
Suma y resta en binario puro.

Bit A	Op.	Bit B	Res	Acarr
1	+	1	0	1
1	+	0	1	0
0	+	1	1	0
0	+	0	0	0
1	-	1	0	0
1	-	0	1	0
0	-	1	1	1
0	-	0	0	0

Al desarrollar las reglas de sumar con signo observaremos que éstas son las correspondientes al binario puro, con algún retoque. Y que de los cuatro códigos, sólo los basados en el complemento son *espontáneos* respecto de la suma con signo.

2.4.1. Desborde u Overflow

Al sumar dos números es muy importante controlar si el resultado obtenido supera el rango asociado al número de bits utilizados, es decir, comprobar que el resultado puede codificarse con los bits disponibles. Esta situación, denominada desbordamiento u *overflow*, no es evitable toda vez que se ha elegido un número de bits, pero sí puede ser detectada y tenida en cuenta. Una cuidadosa elección del número de bits a utilizar en la codificación es determinante para la calidad del sistema digital -de qué nos vale un sistema que opera, si sus resultados no son utilizables-.

A continuación, describiremos las reglas de la suma de números con signo en binario con C-1 y con C-2, basándonos en las reglas de la suma en binario puro, así cada vez que se hable de sumar se presupondrá que se trata de una suma según las reglas de la tabla 2-9.

2.4.2. Suma utilizando el binario con complemento a 1

Los pasos para obtener la suma de dos números con signo y mismo número de bits representados mediante binario con C-1 son:

- Sumar ambos números según la regla aritmética de la suma binaria, propagándose el acarreo entre bits.
- Recircular el acarreo generado por el último bit, es decir, sumar dicho acarreo desde el primer bit, aunque el acarreo sea cero.
- El resultado es correcto, siempre que no haya desborde.

2.4.2.1. Desbordamiento

El desbordamiento se detecta cuando el acarreo (llevada) generado y el recibido por el último bit son distintos. La ecuación booleana que describe el desbordamiento es:

$$D = C_i \oplus C_{i-1}, \qquad (2.6)$$

donde C_i es el acarreo generado por el bit i (último) y C_{i-1} es el acarreo generado por el anteúltimo bit y $\oplus$ es la operación suma exclusiva vista en el capítulo siguiente.

Antes de presentar distintos ejemplos, debemos recordar que al simular las sumas debemos cumplir todos los pasos, sin desechar ninguno por inútil. Por ejemplo, no debemos dejar de recircular el acarreo final porque sea cero.

Ejemplo 2-16

Completar y comprobar las siguientes sumas en binario con C-1. Los números en cursiva son los acarreos, en negrita el acarreo final (consideraremos que el primer acarreo es cero).

00000		*000110*	
0110	(+6)	01001	(+9)
0001	(+1)	00011	(+3)
00111		**0**01100	
+ *0*		+ *0*	
0111	(+7)	01100	(+12)
000000		*1000*	
01000	(+8)	101	(-2)
10011	(-12)	110	(-1)
011011		**1**011	
+ *0*		+ *1*	
11011	(-4)	100	(-3)

Cabe destacar el último caso, (-2)+(-1). En principio parece que hay desborde, pues 1 es distinto de 0, pero al recircular el acarreo ambos acarreos se convierten en cero. Es decir, durante la primera suma se indicaría desbordamiento, situación que desaparecería al concluir la suma. Por tanto, debemos concluir que el estudio del desbordamiento se debe completar al acabar la recirculación del acarreo, o que dicho estudio no es realista hasta que finaliza al completo la suma.

Ejemplo 2-17

Completar y comprobar las siguientes sumas en binario con C-1 en las que se producen situaciones de desborde.

01000		*01100*	
0110	(+6)	0110	(+6)
+ 0100	(+4)	+ 0010	(+2)
~~0~~1010		~~0~~1000	
+ *0*		+ *0*	
1010	(-5)	1000	(-7)
$C_2 = 1$ y $C_3 = 0$		$C_2 = 1$ y $C_3 = 0$	
Desborde		**Desborde**	

10000		*100000*	
1001	(-6)	10000	(-15)
+ 1000	(-7)	+ 1110	(-1)
~~1~~0001		~~1~~01110	
+ *1*		+ *1*	
0010	(+2)	01111	(+15)
$C_2 = 0$ y $C_3 = 1$		$C_2 = 0$ y $C_3 = 1$	
Desborde		**Desborde**	

El problema del rango de los números a representar y a operar debe ser resuelto meticulosamente en cada caso, lo contrario puede invalidar todo el diseño.

2.4.3. Suma utilizando el binario con complemento a 2

Los pasos para obtener la suma de dos números binarios con signo y mismo número de bits representados en binario con C-2 son:

- Sumar ambos números según la regla aritmética de la suma binaria, propagándose el acarreo entre bits.
- Despreciar el acarreo final, no recircularlo ni considerarlo parte del resultado.
- El resultado es correcto, siempre que no haya desborde.

2.4.3.1. Desbordamiento

El desbordamiento se detecta cuando el acarreo (llevada) generado y el recibido por el último bit son distintos, igual que en el caso de binario y C-1. La ecuación booleana que describe el desbordamiento es:

$$D = C_i \oplus C_{i-1}, \qquad (2.7)$$

donde C_i es el acarreo generado por el bit i (último) y C_{i-1} es el acarreo generado por el penúltimo bit.

Ejemplo 2-18

Completar y comprobar las siguientes sumas en binario con C-2. Los números en cursiva son los acarreos (consideramos que el primer acarreo es cero).

00000		*000110*	
0110	(+6)	01001	(+9)
0001	(+1)	00011	(+3)
~~0~~0111	(+7)	~~0~~01100	(+12)

000000		*1100*	
01000	(+8)	110	(-2)
10011	(-13)	111	(-1)
~~0~~11011	(-5)	~~1~~101	(-3)

En este caso, y en comparación con el C-1, la suma (-1) + (-2) no produce transitoriamente ninguna situación de desbordamiento.

Los siguientes ejemplos describen situaciones de desborde, recordemos que para cuatro bits el rango es de -8 a +7, siendo de -16 a +15 para cinco bits.

01000		*01100*	
0110	(+6)	0110	(+6)
+ 0100	(+4)	+ 0010	(+2)
~~0~~1010	(-6)	~~0~~1000	(-8)
$C_2 = 1$ y $C_3 = 0$		$C_2 = 1$ y $C_3 = 0$	
Desborde		**Desborde**	

10000		*111110*	
1001	(-7)	10001	(-15)
+ 1000	(-8)	+ 11111	(-1)
~~1~~0001	(+1)	~~1~~10000	(-16)
$C_2 = 0$ y $C_3 = 1$		$C_3 = 1$ y $C_4 = 1$	
Desborde		**No hay Desborde**	

La representación con C-2 es más cómoda que con C-1 respecto de la suma, ya que no hay que recircular el acarreo final. Didácticamente ambas representaciones son idénticas, incluso es preferible el C-1 por su fácil lectura, pero en la práctica la más utilizada es el binario con C-2.

Los números representados con signo y magnitud o con exceso son susceptibles de ser sumados siguiendo determinados pasos, que serán descritos en el capítulo 6.

2.4.4. Resta de números binarios con signo

Al igual que en el caso de la suma, la resta se podría obtener aplicando las reglas de la tabla de la resta. Ahora bien, existe otro planteamiento más utilizado: algebraicamente reescribimos la resta como una suma con el sustraendo cambiado de signo, $A - B = A + (-B)$, donde tanto A como B son números binarios con signo.

El diseño digital prefiere esta segunda opción, pues aunque necesita de la fase de negación, no es necesario implementar en hardware la tabla de la resta. Por tanto, restar es negar y sumar, y negar es complementar, ya sea a 1 o a 2. La resta de dos números binarios con signo, ya sea en C-1 o en C-2, se obtiene:

- Se obtiene el complemento correspondiente del sustraendo.
- Se suman el minuendo y el sustraendo complementado, según el criterio de la suma con el complemento correspondiente.
- El resultado obtenido es correcto, siempre que no haya desborde.

La tabla 2-10 compara los distintos modos de sumar y restar según trabajemos con el complemento a 1 o a 2.

Tabla 2-10
Suma y resta en binario puro.

	Binario con C-1	Binario con C-2
Suma	Sumar los sumandos, recirculando el acarreo final.	Sumar ambos sumandos, despreciando el acarreo final.
Negación	Complementar a 1.	Complementar a 2.
Resta	Negar el sustraendo y aplicar la regla de la suma con C-1.	Negar el sustraendo y aplicar la regla de la suma con C-2.
Desborde	Cuando el acarreo recibido y el generado por el último bit sean distintos.	Cuando el acarreo recibido y el generado por el último bit sean distintos.

Ejemplo 2-19

Completar y comprobar las siguientes restas en binario con complemento a 1.

```
                 11000                                00000
  (+5)  0101     0101      (+5)      (+4)  0100       0100       (+4)
 -(+3)  0011   + 1100  +   (-3)      -(-3) 1100     + 0011   +   (+3)
                10001                                00111
               +    1                               +    0
                 0010      (+2)                       0111       (+7)

                 00100                                10000
  (-5)  1010     1010      (-5)      (-5)  1010       1010       (-5)
 -(-2)  1101   + 0010  +   (+2)      -(+3) 0011     + 1100   +   (-3)
                01100                                10110
               +    0                               +    1
                 1100      (-3)                       0111       (+7)
```

$C_2=0$ y $C_3=1$

Desborde

Ejemplo 2-20

Completar y comprobar las siguientes restas en binario con complemento a 2.

		11010		
(+5)	0101	0101		(+5)
- (+3)	0011	+ 1101	+	(-3)
		~~1~~0010		(+2)
		00110		
(-5)	1011	1011		(-5)
- (-3)	1101	+ 0011	+	(+3)
		~~0~~1110		(-2)
		10000		
(+4)	0100	0100		(+4)
- (-4)	1100	+ 0100	+	(+4)
		~~0~~1000		(-8)
		C_2=0 y C_3=1		
		Desborde		
		11110		
(-5)	1011	1011		(-5)
- (+3)	0011	+ 1101	+	(-3)
		~~1~~1000		(-8)

2.5. Representación de números reales

En las secciones anteriores hemos anotado cómo representar números enteros, con y sin signo. Resulta lógico pensar que también existirá una manera de representar magnitudes reales; con decimales.

La representación de números reales queda fuera de los objetivos de este libro, que son básicos. El lector interesado podrá estudiar dicha representación en gran cantidad de libros, sobre todo en aquellos enfocados hacia la arquitectura de computadores.

2.6. Codificación de magnitudes en notación binaria

En los apartados anteriores hemos descrito el sistema binario con sus diversas variantes; en este punto describiremos los distintos códigos que se basan en el sistema binario, y que en el caso del código binario puro es idéntico a él.

El código binario puro es de gran importancia teórica, sin embargo los computadores exigen del código utilizado diversas propiedades que el binario puro no satisface al completo, es más, no existe un único código que aventaje claramente a los demás. A continuación, describiremos las distintas estrategias de codificación, los códigos que de éstas se derivan y sus ventajas y desventajas.

2.6.1. Aspectos de la codificación en binario

Antes de abordar la descripción de los distintos códigos, es necesario definir una serie de conceptos y propiedades.

Un bit es la menor cantidad de información binaria, constituido por un único elemento de información, cuyo valor es 0 o 1. Con k bits, pueden representarse 2^k números como máximo, por ejemplo, con cuatro bits, tenemos $2^4 = 16$ posibilidades. Esta cuestión también es planteable a la inversa: ¿cuántos bits son necesarios para representar M números?

Por ejemplo, para representar los 10 estados del sistema decimal hacen falta $K = \log_2 M = \log_2 10 = 3{,}2$ bits; pero el número de bits ha de ser entero y por tanto necesitamos 4. Para el alfabeto romano de 26 símbolos necesitaríamos $k = \log_2 26 = 4{,}7$ bits, luego 5 bits. Esta redundancia es inevitable, de 0,7 bits redundantes o 6 combinaciones redundantes en el primer caso, y de 0,3 bits redundantes o 6 combinaciones redundantes en el segundo caso. En principio la redundancia parece no deseable, pero es una característica de los códigos que detectan e incluso corrigen errores.

Los códigos binarios pueden clasificarse según sea la conversión decimal-binario, y viceversa. Así, un código puede ser:

- Códigos de Palabra: codifican en binario el número decimal como un todo, suelen ser de longitud variable. Intuitivamente, digamos que a un número decimal de n dígitos, se le asocia un esfuerzo de grado n para codificarlo; es más fácil codificar en binario puro el 8 que el 3.075. Este tipo de códigos no suelen utilizarse en la práctica, sus principales representantes son el binario puro y el código Gray.
- Códigos BCD (Binary Code Digit): asocian a cada dígito decimal una combinación de dígitos binarios. En este caso codificar un número de n dígitos supone codificar n veces un dígito decimal; este mecanismo es más sencillo que el correspondiente a códigos de palabra. Mediante una tabla o regla asociamos a cada dígito decimal una combinación de bits. Los códigos BCD son los más utilizados y son siempre redundantes. Los más conocidos son el BCD 8-4-2-1, el BCD-XS3 y el Gray-XS3.

Ejemplo 2-21

Codificación del número 18, en códigos de palabra y BCD.

Decimal	B. Puro	Binario Gray
18	10010	11011

Decimal	BCD Puro	BCD XS3	Gray XS3
18	00011000	01001011	01101110

2.6.1.1. Características de los códigos

Como ya hemos dicho, los códigos son variados y diversos, y por tanto se hace necesario caracterizarlos para poder seleccionar en cada caso el más adecuado. Sin ánimo excesivamente teórico pasamos a describir algunas de las principales características y propiedades de los códigos, que principalmente se aplican a los códigos BCD.

Cada combinación de dígitos binarios asociada a un dígito decimal se denomina ***palabra de código***. Su ***longitud*** ln indica el número de bits de cada palabra, y su ***peso*** g se corresponde con el número de 1's de la palabra, así la palabra 0100111 es de longitud siete y de peso cuatro, ln = 7 y g = 4.

La ***palabra nula*** es aquella con todo ceros, g = 0, y la ***palabra unidad*** es aquella con todo unos, ln = g. No es deseable que estas palabras pertenezcan al código, pues errores mecánicos o eléctricos pueden generar estas palabras, introduciendo ambigüedad: ¿error o información?

Si todas las palabras tienen la misma longitud, se dice que el ***código*** es ***uniforme***; éste es el caso más común. La cantidad de combinaciones binarias aceptadas por el código es M. Si M es igual a 2^{ln}, entonces el ***código*** es ***completo*** o sin redundancia; lo común es que los códigos BCD sean incompletos.

Distancia

Dadas dos palabras de un código completo, su distancia d indica el número de diferencias entre ambas combinaciones. Así, en p1 = 0100 y p2 = 0010, d(p1,p2) = 2.

Se cumplen las siguientes relaciones respecto de la distancia:

$$d(p1,p1) = 0$$

$$d(p1,p2) = d(p2,p1)$$

$$d(p1,\text{palabra } 0) = g(p1)$$

$$d(p1,\text{palabra } 1) = ln-g(p1)$$

$$\text{mod2}(d(p1,p2)) = \text{mod2}(g(p1)+g(p2))$$

$$d(p1,p2) = g(p1\oplus p2)$$

Ejemplo 2-22

Verificación de las relaciones para p1=0100 y p2=0010.

d(p1,p2) = 2 (par)
g(p1) = 1
g(p2) = 1
g(p1) + g(p2) = 2 (par)
d(p1,0000) = 1
d(p1,1111) = 4-1 = 3

La ***distancia mínima***, o simplemente distancia, es la menor que puede haber entre dos palabras cualesquiera de la tabla de código. En un código completo su distancia siempre es 1, sólo en los códigos con redundancia la distancia puede ser mayor que 1.

2.6.2. Códigos de palabra

Ya se ha dicho que un código de palabra es aquel que para convertir un número decimal a binario trabaja con todo el número -mientras BCD lo divide en dígitos-. Su principal característica es su longitud variable: números con la misma cantidad de dígitos decimales tienen distinta cantidad de bits, y viceversa.

Ejemplo 2-23

Codificar en binario puro 1, 4, 8 y 14.

1 = 1 (1 bit) 4 = 100 (3 bits)
8 = 1000 (4 bits) 14 = 1110 (4 bits)

Los dos códigos de palabra principales son: el binario puro y el Gray. El primero es el explicado para el sistema binario, y el Gray será comentado en el punto 2.6.4 con los códigos de paso simple.

2.6.3. Códigos BCD

No son los únicos existentes, pero sí son los más utilizados. Asignan a cada dígito decimal una combinación de dígitos binarios, y dicha correspondencia queda resumida en la tabla de código. Los códigos según la forma de obtener las tablas pueden ser de dos tipos: ponderados u ordenados.

Un código es ponderado si a cada uno de los n dígitos binarios se le puede asignar un peso, y por tanto entre el dígito decimal y los binarios se puede establecer una relación:

$$\sum_{i=0}^{n-1} P_i \times B_i \qquad (2.8)$$

donde P_i es el peso i-ésimo y B es el bit i-ésimo

La distribución de pesos más popular es 8-4-2-1 (2^3, 2^2, 2^1, 2^0), pero también puede ser 5-4-2-1, o 2-4-2-1, etc.

Ejemplo 2-24

Interpretar 1001 según los pesos 8-4-2-1, 5-4-2-1 y 2-4-2-1.

1 x 8 + 0 x 4 + 0 x 2 + 1 x 1 = 9
1 x 5 + 0 x 4 + 0 x 2 + 1 x 1 = 6
1 x 2 + 0 x 4 + 0 x 2 + 1 x 1 = 3

Existe un gran número de códigos ponderados, pero todos ellos deben cumplir las siguientes normas básicas:

- El peso asignado al lugar i-ésimo no puede superar en más de uno a la suma de los pesos de las anteriores posiciones, en caso contrario se producirían "*agujeros*" en el código: $P_i \leq 1 + P_{i-1} + P_{i-2} + ... + P_0$. Por ejemplo, el código BCD 8-4-2-1 es válido, no así el posible código BCD 9-4-2-1.
- La suma de todos los pesos ha de ser al menos 9;
$P_0 + P_1 + P_2 + ... + P_{l-1} \geq 9$. Por ejemplo, el código BCD 2-4-2-1.
- Un peso sólo puede aparecer una vez en la distribución de pesos para evitar ambigüedades. Por ejemplo, la ambigüedad se da en el código BCD 2-4-2-1, ya que el peso 2 aparece duplicado. Así, el número 6 tiene una codificación ambigua, ya que puede codificarse como 1100 o 0110. Esta ambigüedad queda resuelta arbitrariamente en la tabla 2-11 , y así el 6 queda codificado como 1100.

Todos los códigos que no son ponderados son ordenados, en cualquier caso todos ellos quedan definidos mediante su tabla de código, y algunos de ellos por su modo de obtención. En los párrafos siguientes describiremos algunos códigos BCD.

Para representar 10 símbolos (0-9), cuatro bits es el menor número de bits necesarios. De las 16 posibles combinaciones sólo son usadas 10, las 6 restantes se llaman pseudodecimales o pseudotetradas.

Antes de describir algunos códigos genéricos de 4 bits, responderemos a la cuestión de cuántos códigos existen si hay que representar P caracteres con C posibles combinaciones:

$$\text{Número Códigos} = \binom{C}{P} \cdot P! = \frac{C!}{(C-P)!P!} \text{ x } P! = \frac{C!}{(C-P)!} \qquad (2.9)$$

Ejemplo 2-25

¿Cuántos códigos de diez palabras se pueden obtener con cuatro bits?

$$\text{Códigos de 4 bits} = \binom{14}{10} \bullet 10! = \frac{16!}{6!} = 2{,}9 \text{ x } 10^{10}$$

La cantidad de posibles códigos es enorme, varios de ellos aparecen en la tabla 2-11, y cada uno tiene sus ventajas y sus desventajas.

Tabla 2-11
Códigos BCD más comunes de cuatro bits.

B1	B2	B3	B4	B.Puro	8-4-2-1	Aiken	XS3	Salto al 2	Salto al 8	4-2-2-1	5-4-2-1	5-2-2-1	5-3-1-1	White
0	0	0	0	0	0	0		0	0	0	0	0	0	0
0	0	0	1	1	1	1		1	1	1	1	1	1	1
0	0	1	0	2	2	2			2	2	2	2		
0	0	1	1	3	3	3	0		3	3	3	3	2	2
0	1	0	0	4	4	4	1		4		4		3	
0	1	0	1	5	5		2		5				4	3
0	1	1	0	6	6		3		6	4		4		
0	1	1	1	7	7		4		7	5				4
1	0	0	0	8	8		5	2			5	5	5	5
1	0	0	1	9	9		6	3			6	6	6	6
1	0	1	0	10			7	4			7	7		
1	0	1	1	11		5	8	5			8	8	7	7
1	1	0	0	12		6	9	6		6	9		8	
1	1	0	1	13		7		7		7			9	8
1	1	1	0	14		8		8	8	8		9		
1	1	1	1	15		9		9	9	9				9

2.6.3.1. Código 8-4-2-1 o BCD puro

El código 8-4-2-1 es el más popular de los ponderados y se usa mucho en representación y suma. Utiliza las 10 primeras combinaciones, es asimétrico, utiliza la palabra nula y la obtención de su complemento a 9 no es sencilla.

2.6.3.2. Código Aiken

Si observamos el código Aiken o 2-4-2-1, el código es ponderado y simétrico y el complemento a 9 se obtiene por inversión de los bits. Tiene como inconveniente su ambigüedad y el uso de las palabras 0 y 1.

2.6.3.3. Código BCD-XS3

El código BCD-XS3 (exceso a 3) de Stibitz es no ponderado, simétrico y su complemento a 9 es sencillo, al igual que el de Aiken, pero, frente a éste, no contiene las palabras 1 o 0. Aunque la tabla define al XS3 podemos dar sus reglas de conversión:

- Decimal a XS3. Se suma tres al decimal original y se codifica como si fuera BCD 8-4-2-1.

$$\text{BCD - XS3 (D)} = \text{8-4-2-1 (D + 3)} \qquad (2.10)$$

• XS3 a Decimal. Aplicando la siguiente expresión:

$$\text{Dec} = \left(\sum_{i=0}^{l-1} P_i \cdot B^i \right) - 3 \; ; \; (P_i=8,4,2,1) \qquad (2.11)$$

Ejemplo 2-26

Codificar 3 en BCD-XS3 y decodificar 1100 del XS3.

XS3(3) = 8-4-2-1 (3 + 3) = 8-4-2-1 (6) = 0110

Dec. = (1 x 8 + 1 x 4 + 0 x 2 + 0 x 1) - 3 = 12 - 3 = 9

La aritmética del XS3 es comparable a la correspondiente al 8-4-2-1, incluso mejor, como veremos en el capítulo 6.

2.6.3.4. Otros códigos

Veamos algún código BCD más por pura curiosidad.

Las formas especiales del código 2-4-2-1 con salto a 2 y a 8 son utilizadas en conversores y contadores unidireccionales, aunque no resulten útiles en absoluto en otras situaciones.

El código 5-4-2-1 es simétrico, con agrupamiento de 5, su ponderación no es ambigua, utiliza la palabra nula pero no la unidad, sin embargo la obtención del complemento es complicada. Los códigos 5-2-2-1 , 5-3-1-1 y el de White también son simétricos con agrupamiento de 5, y además el de White permite construir sumadores muy sencillos.

La tabla 2-12 muestra los códigos descritos anteriormente ordenados en base al código decimal.

Tabla 2-12

Códigos BCD más comunes de cuatro bits.

Decimal	8-4-2-1	AIKEN	Exceso 3 XS3	Salto al 2	Salto al 8
Peso	8 4 2 1	2 4 2 1	NO	2 4 2 1	2 4 2 1
0	0 0 0 0	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 1 0 0	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0	0 1 0 1	1 0 0 0	0 0 1 0
3	0 0 1 1	0 0 1 1	0 1 1 0	1 0 0 1	0 0 1 1
4	0 1 0 0	0 1 0 0	0 1 1 1	1 0 1 0	0 1 0 0
5	0 1 0 1	1 0 1 1	1 0 0 0	1 0 1 1	0 1 0 1
6	0 1 1 0	1 1 0 0	1 0 0 1	1 1 0 0	0 1 1 0
7	0 1 1 1	1 1 0 1	1 0 1 0	1 1 0 1	0 1 1 1
8	1 0 0 0	1 1 1 0	1 0 1 1	1 1 1 0	1 1 1 0
9	1 0 0 1	1 1 1 1	1 1 0 0	1 1 1 1	1 1 1 1

Tabla 2-12 (cont.)
Códigos BCD más comunes de cuatro bits.

	4-2-2-1				5-4-2-1				5-2-2-1				5-3-1-1				WHITE			
Peso	**4**	**2**	**2**	**1**	**5**	**4**	**2**	**1**	**5**	**2**	**2**	**1**	**5**	**3**	**1**	**1**	**5**	**2**	**1**	**1**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	1	1
3	0	0	1	1	0	0	1	1	0	0	1	1	0	1	0	0	0	1	0	1
4	0	1	1	0	0	1	0	0	0	1	1	0	0	1	0	1	0	1	1	1
5	0	1	1	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
6	1	1	0	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
7	1	1	0	1	1	0	1	0	1	0	1	0	1	0	1	1	1	0	1	1
8	1	1	1	0	1	0	1	1	1	0	1	1	1	1	0	0	1	1	0	1
9	1	1	1	1	1	1	0	0	1	1	1	0	1	1	0	1	1	1	1	1

Las tablas 2-11 y 2-12 ya han dado idea de la variedad de códigos BCD que hay para cuatro bits. El número se amplía si pensamos en 5 o más bits, pero estos códigos quedan claramente fuera de este libro.

2.6.4. Códigos de palabra de paso simple

En los procesos automatizados, ciertas señales son medidas y digitalizadas para controlar digitalmente el proceso. Estas medidas pueden representar valores correlativos en el tiempo, por ejemplo un disco codificado de ángulos genera 0°, 10°, 20°, ..., 360°.

Lo mismo ocurre con ciertas medidas de longitud. Estos discos codificadores son de tipo mecánico u óptico, y así al pasar de un valor a otro los mecanismos cambian de valor. Si se dan múltiples cambios al pasar de un valor al siguiente se pueden producir errores o lecturas falsas, por ejemplo al pasar de 7 (0111) a 8 (1000) se producen cuatro cambios (d = 4); esta situación puede traer problemas. Además, estos saltos suponen un esfuerzo mecánico o de otro tipo, lo que conlleva una rápida degradación del mecanismo. Para evitar esta situación se desarrollan los códigos de paso simple, aquellos cuya distancia es 1 entre combinaciones consecutivas, de tal forma que se reducen las posibilidades de error y el esfuerzo del mecanismo.

El principal código de paso simple es el código GRAY, que está directamente relacionado con el binario puro. Un número en binario puro b3 b2 b1 b0 (o cualquier secuencia binaria) y su correspondiente GRAY g3 g2 g1 g0, se relacionan como sigue:

$g3 = b3 \qquad b3 = g3$

$g2 = b3 \oplus b2 \qquad b2 = g3 \oplus g2$

$g1 = b2 \oplus b1 \qquad b1 = g3 \oplus g2 \oplus g1$

$g0 = b1 \oplus b0 \qquad b0 = g3 \oplus g2 \oplus g1 \oplus g0$

Ejemplo 2-27

Convertir 10100 a Gray y 0011 a binario.

$B_4B_3B_2B_1B_0 = 10100$	$G_3G_2G_1G_0 = 0011$
$G_4 = 0$	$B_3 = 0$
$G_3 = 1 \oplus 0 = 1$	$B_2 = 0 \oplus 0 = 0$
$G_2 = 0 \oplus 1 = 1$	$B_1 = 0 \oplus 0 \oplus 1 = 1$
$G_1 = 1 \oplus 0 = 1$	$B_0 = 0 \oplus 0 \oplus 1 \oplus 1 = 0$
$G_0 = 0 \oplus 0 = 0$	$B_3B_2B_1B_0 = 0010$
$G_4G_3G_2G_1G_0 = 01110$	

La tabla 2-13 nos muestra la relación entre el código GRAY y el correspondiente binario puro para los números de 1 a 18. El GRAY es un código de palabra completa, no es BCD, es no ponderado y presenta una simetría especular, donde los bits de menor peso se reflejan respecto de la línea de simetría, de ahí que este código sea conocido como código especular.

Tabla 2-13
Binario puro y Gray de paso simple.

	Código de GRAY					Código binario puro				
			no			16	8	4	2	1
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	1	0	0	0	1	0
3	0	0	0	1	0	0	0	0	1	1
4	0	0	1	1	0	0	0	1	0	0
5	0	0	1	1	1	0	0	1	0	1
6	0	0	1	0	1	0	0	1	1	0
7	0	0	1	0	0	0	0	1	1	1
8	0	1	1	0	0	0	1	0	0	0
9	0	1	1	0	1	0	1	0	0	1
10	0	1	1	1	1	0	1	0	1	0
11	0	1	1	1	0	0	1	0	1	1
12	0	1	0	1	0	0	1	1	0	0
13	0	1	0	1	1	0	1	1	0	1
14	0	1	0	0	1	0	1	1	1	0
15	0	1	0	0	0	0	1	1	1	1
16	1	1	0	0	0	1	0	0	0	0
17	1	1	0	0	1	1	0	0	0	1
18	1	1	0	1	1	1	0	0	1	0

Si bien el GRAY es el más conocido teóricamente, en la practica no es útil, por no ser BCD. Se han desarrollado gran número de códigos de paso simple tipo BCD. La tabla 2-14 da idea de esta variedad.

Tabla 2-14
Códigos BCD de paso simple.

	GRAY	GLIXON	O'BRIEN I	O'BRIEN II	TOMPKINS I	TOMPKINS II	XS3-GRAY
0	0000	0000	0000	0001	0000	0010	0010
1	0001	0001	0001	0011	0001	0011	0110
2	0011	0011	0011	0010	0011	0111	0111
3	0010	0010	0010	0110	0010	0101	0101
4	0110	0110	0110	0100	0110	0100	0100
5	0111	0111	1110	1100	1110	1100	1100
6	0101	0101	1010	1110	1111	1101	1101
7	0100	0100	1011	1010	1101	1001	1111
8	1100	1100	1001	1011	1100	1011	1110
9	1101	1000	1000	1001	1000	1010	1010

2.6.5. Códigos BCD con signo

Existen muy diversas formas de representar números en BCD con signo, todas válidas cualquiera que sea el BCD elegido, pero de todas ellas tres son las más empleadas:

- Bit de signo. En este caso se añade delante de la magnitud un bit de signo, cero para los positivos y uno para los negativos, al igual que en binario con magnitud y bit de signo. Estos 0 y 1 bien pueden ser un bit o todo un dígito BCD, por ejemplo 0000 o 0001.
- Complemento a 10. El número negativo se codifica en BCD como el complemento a 10 del correspondiente positivo. El complemento a 10 es al BCD lo que el C-2 es al binario puro; el complemento a la base.
- Complemento a 9. El número negativo se codifica en BCD como el complemento a 9 del correspondiente positivo. El complemento a 9 es al BCD lo que el C-1 es al binario puro: el complemento a la base menos 1.

Ejemplo 2-28

Codificar -12 en BCD-XS3 según las tres técnicas anteriores.

Bit de signo -12 = 1 0100 0101
Compl. a 10 -12 = C-10(12) = 88 = 1011 1011
Compl. a 9 -12 = C-9(12) = 87 = 1011 1010

Al representar números con signo es cuando se entiende el valor añadido que tiene un código si su complemento a 9 es una operación sencilla, por ejemplo el XS3.

En cuanto a la suma y la resta utilizando códigos BCD, éstas serán planteadas en el capítulo 6 dedicado a los elementos aritméticos.

2.7. Códigos detectores y correctores de error

Existe un tipo de código capaz de detectar, e incluso corregir, los errores producidos en una transmisión. Es decir, la información enviada por el emisor está codificada de tal manera que el receptor es capaz de decidir si lo recibido es correcto o contiene algún error. Este campo de investigación está ahora mismo en auge para establecer comunicaciones seguras de cualquier tipo.

Los códigos desarrollados y en desarrollo son muchos, y se basan tanto en ideas sencillas como en desarrollos muy complejos. Algunos de los códigos de error son:

- Bit de paridad.
- Código Hamming.
- Códigos CRC.
- Códigos con checksum.

Los dos primeros son los más comunes. El primero sólo es capaz de decir si la información recibida es correcta o no, mientras que el código de Hammimng es capaz de detectar el error e incluso corregirlo, aumentando claramente la calidad de la transmisión. Ambos códigos tienen como base el añadir alguno o algunos bits más a la información a transmitir, y además en el caso de Hamming se busca aumentar la distancia del código, haciendo que d sea igual a 3, 5, etc.

En este libro presentaremos exclusivamente los códigos basados en el bit de paridad, que serán explicados en el tema 4.

2.8. Resumen

Los sistemas digitales utilizan el sistema binario para representar y operar con magnitudes.

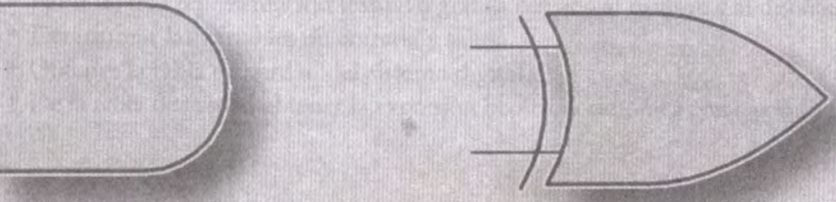
Existe una gran variedad de códigos binarios, cada uno con sus ventajas y desventajas. De todos ellos, los más utilizados son el binario puro y el BCD para magnitudes sin signo, y el binario puro con C-1 y C-2 para las magnitudes con signo.

Antes de pasar a los siguientes capítulos el lector debe ser capaz de leer cualquier secuencia binaria, y de escribir cualquier secuencia decimal, sobre todo en los códigos básicos. En el resto del libro será más importante el sistema binario que el decimal.

CAPÍTULO

3

ÁLGEBRA DE BOOLE



3.1. Introducción

En el siglo pasado, el filósofo y matemático ingles George Boole desarrolló una teoría matemática revolucionaria cuya principal característica era el manejo de variables con sólo dos valores posibles: verdadero y falso, 1 y 0. El resultado publicado en 1854, *"An Investigation of the laws of thought"*, ofrece un marco de manipulación simbólica, de tipo algebraico, del razonamiento lógico, es decir, presenta un modelo matemático para la lógica proposicional.

En los años treinta, el ingeniero electrónico del MIT, Claude E. Shannon, en los laboratorios Bell, afirma que el álgebra de Boole constituye una formalización algebraica apropiada para el estudio de los circuitos electrónicos de conmutación con sólo dos estados posibles (*"A symbolic Analysis of Relay and Switching Circuits"*, 1938).

Desde esos años treinta la electrónica digital ha alcanzado un gran desarrollo, tanto en la complejidad como en la diversidad de sus aplicaciones, y ha contemplado cómo muy diversas tecnologías han servido de base para su implementación. En todos estos años el álgebra de Boole ha sido la herramienta matemática en el análisis y diseño de sistemas digitales, independientemente del campo de aplicación y de la tecnología aplicada.

A continuación vamos a presentar los fundamentos y reglas del álgebra de Boole, para ello no entraremos en disertaciones excesivamente matemáticas, centrándonos preferentemente en los resultados y su utilidad. Previamente a la descripción y formalización del álgebra de Boole es necesario conocer las distintas formas de representar sistemas digitales.

3.2. Representación de sistemas digitales

Todo sistema digital puede ser representado o expresado de muy distintas formas. Cada una de estas técnicas destaca determinado aspecto del sistema digital, teniendo ventajas respecto de las otras, y viceversa. Por ejemplo, la representación utilizada puede ser:

- Cercana a la expresión humana.
- Fácil de obtener.
- Cómoda para su manipulación booleana.
- Cómoda para su implementación.
- Favorable a la simplificación.
- Favorable al análisis temporal.

Todas las técnicas anteriores deben expresar siempre lo mismo, ya que el sistema no varía, y por tanto es lógico pensar que se pueda pasar de una a otra.

Para diseñar un circuito combinacional a nivel de bit sencillo la secuencia a seguir puede ser ésta:

- Partir de una representación textual o gráfica cercana al usuario y al diseñador.
- Determinar las variables de entrada y salida.
- Obtener la tabla de verdad del sistema digital.
- De la tabla de verdad obtener la expresión booleana canónica correspondiente.

- De los diagramas V-K derivados de la tabla de verdad obtener las expresiones booleanas simplificadas.
- Reescribir las expresiones booleanas simplificadas en base a un solo tipo de puertas lógicas.
- Dibujar el circuito lógico correspondiente a las expresiones simplificadas.
- Redibujar el anterior esquema digital según sean los circuitos integrados elegidos por su implementación.
- Implementar el circuito.

Algunos de estos pasos quedarán aclarados o intuidos en este tema, otros lo serán más adelante.

3.2.1. Tabla de verdad

La tabla de verdad es la representación más común del comportamiento de un sistema digital binario. En la tabla de verdad queda expresado de forma completa y sin ambigüedades lo que ocurre en la salida del sistema digital en función de los valores de las variables de entrada.

Se divide en dos partes: entrada y salida.

- La entrada se coloca en la parte izquierda, y se compone de tantas columnas como variables de entrada haya. Tendrá 2^n filas, donde n es el numero de variables. Cada fila de la entrada es una combinación distinta de las otras filas, y debe entenderse como una combinación de valores de las variables de entrada que se propone al sistema digital para obtener su salida.
- La salida se coloca en la derecha. Tiene el mismo numero de filas que la entrada, y tantas columnas como variables de salida. El valor asignado a cada fila de cada columna (0 o 1) se obtiene como respuesta a la siguiente pregunta: ¿Qué valor debe tomar la salida si el valor de la entrada es el representado por su fila, siendo conocido el comportamiento del sistema?

Destaquemos que en la entrada siempre se colocarán 2^n filas, si alguna de las combinaciones de las variables de entrada no tiene sentido lógico, no quiere decir que deba ser eliminada de la tabla de verdad; si lo es, la tabla de verdad quedará incompleta y el resultado podrá no ser correcto. La razón estriba en que existen dos conceptos claramente diferenciados que aparecerán continuamente en el desarrollo del texto: información lógica *vs* información física. Es muy distinto que una situación no pueda darse por razones lógicas a que no se produzca la situación electrónica que lo representa. Por ejemplo, pensemos en dos pulsadores para indicar *arranque* y *paro*; no se pueden dar los dos hechos a la vez, pero sí se pueden activar ambos pulsadores a la vez, aunque no se deba.

3.3. Definición del álgebra de Boole

Existen diversas formas para definir el álgebra de Boole, en nuestro caso vamos a centrarnos en el álgebra de Boole bivalente, que queda definida por las siguientes reglas (la discusión de si se trata de axiomas o postulados concierne a la epistemología, y no a este texto):

1. Para toda variable booleana bivaluada x.

$$x = 0 \text{ o } x = 1.$$

2. Operación complemento o negación: función **NOT**

$$\overline{0} = 1$$
$$\overline{1} = 0$$

3. Operación producto booleano o producto lógico: función **AND**.

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

4. Operación suma booleana o suma lógica: función **OR**.

$$0+0 = 0$$
$$0+1 = 1$$
$$1+0 = 1$$
$$1+1 = 1$$

5. Por convenio, la operación producto lógico es precedente respecto de la suma lógica.

La primera regla define qué es una variable lógica, las tres siguientes definen las tres operaciones básicas del álgebra de Boole (resumidas en la tabla 3-1). Implícitamente indica que el álgebra booleana es cerrada, es decir, que el resultado de operar variables booleanas (+, ·, ó ¯) es una nueva variable booleana.

Tabla 3-1
Operaciones básicas.

x	$\overline{x}$	x	y	x · y	x	y	x + y
0	1	0	0	0	0	0	0
1	0	0	1	0	0	1	1
		1	0	0	1	0	1
		1	1	1	1	1	1

Volvamos a Shannon; su mérito fue que identificó el álgebra de Boole como la herramienta matemática adecuada para el análisis y diseño de sistemas digitales, dotando a esta nueva tecnología de las ventajas que en análisis y diseño proporciona un marco matemático.

3.3.1. Postulados del álgebra de Boole

Si bien bastaría con la anterior definición, también podemos ordenar lo anterior como postulados en la tabla 3-2.

Tabla 3-2
Postulados del álgebra de Boole.

Postulados	Postulados
(P1) $x = 0$ si $x <> 1$	(P1D) $x = 1$ si $x <> 0$
(P2) si $x = 0 \rightarrow \overline{x} = 1$	(P2D) si $x = 1 \rightarrow \overline{x} = 0$
(P3) $0 \cdot 0 = 0$	(P3D) $1 + 1 = 1$
(P4) $1 \cdot 1 = 1$	(P4D) $0 + 0 = 0$
(P5) $1 \cdot 0 = 0 \cdot 1 = 0$	(P5D) $0 + 1 = 1 + 0 = 1$

A continuación presentaremos los principales teoremas del álgebra de Boole, para así describirla mejor y dotarla de aplicabilidad.

Metodológicamente, a la hora de demostrar un teorema se pueden seguir dos caminos; uno abstracto basado en la manipulación simbólica -utilizado masivamente en las demostraciones *normales*-. Otro camino consiste en comprobar la veracidad del teorema para todos los posibles valores de las variables implicadas. Este camino no puede seguirse por ejemplo en el campo de los números reales, ya que su conjunto de valores es infinito, pero sí puede seguirse para variables booleanas ya que sus únicos valores posibles son 0 y 1. Este método de demostración se conoce con el nombre de inducción perfecta (y también ha dado mucho que hablar a los epistemólogos).

En el siguiente apartado, en algunos teoremas utilizaremos el método de manipulación, en otros el método de inducción perfecta y otros quedarán en manos del lector. En cualquier caso, para nosotros no es determinante el conocer y dominar todas las demostraciones, bastará con conocer los teoremas y sus efectos.

3.3.2. Teoremas del álgebra de Boole

A continuación enunciamos y demostramos los principales teoremas del álgebra de Boole (resumidos en la tabla 3.3). Para seguir con la discusión anterior, decir que los cuatro primeros teoremas son enunciados como postulados por otros autores.

Para toda variable booleana se cumplen los siguientes teoremas:

T1. Propiedad conmutativa

$$x + y = y + x \qquad\qquad x \cdot y = y \cdot x$$

Demostración por inducción perfecta

x	y	x + y	y + x
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Ambas columnas resultados son idénticas para todos los posibles valores de x e y, luego ambas expresiones son iguales.

T2. Elemento identidad

$$0 + x = x \quad 1 \cdot x = x$$

T3. Propiedad distributiva

$$x \cdot (y + z) = x \cdot y + x \cdot z$$
$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

T4. Elemento complementario

$$x + \overline{x} = 1 \quad x \cdot \overline{x} = 0$$

T5. Propiedad de idempotencia

$$x + x = x \quad x \cdot x = x$$

T6. Elemento nulo

$$x + 1 = 1 \quad x \cdot 0 = 0$$

Demostración algebraica:

$x + 1 = (x + 1) \cdot 1 =$	(T2)
$= (x + 1)(x + \overline{x}) =$	(T4)
$= (x \cdot x + x \cdot \overline{x} + 1 \cdot x + \overline{x}) =$	(T3)
$= (x + 0 + x + \overline{x}) =$	(T4), (T5) y (T2)
$= (x + x + \overline{x}) =$	(T2)
$= (x + \overline{x}) = 1$	(T5) y (T4)

T7. Ley de convolución

$$\overline{\overline{x}} = x$$

T8. Ley de absorción

$$x + x \cdot y = x \qquad x \cdot (x + y) = x$$

Demostración algebraica:

$x \cdot 1 + x \cdot y =$	(T2)
$= x(1 + y) =$	(T3)
$= x \cdot 1 =$	(T6)
$= x$	(T2)

T9. Propiedad asociativa

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z \qquad x + (y + z) = (x + y) + x$$

T10. Teorema del consenso

$$xy + \overline{x}z = xy + \overline{x}z + yz$$
$$(x + y)(\overline{x} + z) = (x + y)(\overline{x} + z)(y + z)$$

El término $y \cdot z$ es el consenso de los otros dos.

T11. Teorema de Morgan

$$\overline{(x + y)} = \overline{x}\ \overline{y} \qquad \overline{(x \cdot y)} = \overline{x} + \overline{y}$$

Demostremos el teorema de Morgan para dos variables por el método de inducción perfecta.

Tabla 3-3
Teorema de Morgan.

x	y	$f=\overline{x+y}$	$f=\overline{x}\cdot\overline{y}$	$f=\overline{x\cdot y}$	$f=\overline{x}+\overline{y}$
0	0	1	1	1	1
0	1	0	0	1	1
1	0	0	0	1	1
1	1	0	0	0	0

Morgan (realmente es DeMorgan) fue contemporáneo de Boole, precursor junto con él de la lógica matemática. Su libro "*Formal Logic*", apareció el mismo día que el "*Mathematical Analysis of Logic*" de Boole.

El teorema de Morgan es de una gran importancia en la implementación de funciones booleanas, pues enuncia que todo producto puede ser expresado como una suma negada y que toda suma puede ser expresada como un producto negado.

Extensión del teorema de Morgan

$$\overline{(x + y + z)} = \overline{x} \cdot \overline{y} \cdot \overline{z}$$
$$\overline{(x \cdot y \cdot z)} = \overline{x} + \overline{y} + \overline{z}$$
$$\overline{f}(\underline{X}_i,+,\cdot) = (\overline{\underline{X}_i},\cdot,+),$$

donde $\underline{X}_i$ es el vector de variables $x_1,x_2,x_3 \ldots x_n$.y $\overline{\underline{X}_i}$ son dichas variables negadas.

T12. Teorema de Shannon

Para toda función f que relaciona variables booleanas, resulta

$$f(x_1,x_2,\ldots x_i,\ldots xn) = x_i \cdot f(x_1,x_2,\ldots 1,\ldots x_n) + \overline{x_i} \cdot f(x_1, x_2,\ldots 0,\ldots x_n)$$
$$f(x_1,x_2,\ldots x_i,\ldots x_n) = (x_i + f(x_1,x_2,\ldots 0,\ldots x_n)) \cdot (\overline{x_i} + f(x_1, x_2,\ldots 0,\ldots x_n))$$

Por ejemplo, $f = x \cdot y + \overline{x} \cdot z + \overline{y} \cdot \overline{z}$

$$f = x \cdot f(1,y,z) + \overline{x}\, f(0,y,z) = x\,(y + \overline{y}\ \overline{z}) + x\,(z + \overline{y}\ \overline{z})$$

Este teorema es de una gran importancia teórica, como se verá más adelante.

Es interesante comparar el álgebra de Boole con el álgebra de los números reales; se observan diferencias:

- En el álgebra de Boole la suma es distributiva respecto del producto, y viceversa, no así en el álgebra de los números reales.
- En el álgebra de Boole no existe la operación inversa de la + y ·, no existen la resta ni la división.
- En el álgebra de Boole existe la operación complemento, y no en los números reales.
- La propiedad asociativa es un postulado de los números reales, mientras que en álgebra de Boole es un teorema derivado de los postulados.

Tabla 3-3
Teoremas principales del álgebra de Boole.

Teorema		
Conmutativa	$x + y = y + x$	$x \cdot y = y \cdot x$
Identidad	$0 + x = x$	$1 \cdot x = x$
Distributiva	$x \cdot (y + z) = (x \cdot y)+(y \cdot z)$	$x \cdot (y + z) = (x \cdot y)+(x \cdot z)$
Asociativa	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	$x + (y + z) = (x + y) + x$
Complementario	$x + \overline{x} = 1$	$x \cdot \overline{x} = 0$
Idempotencia	$x + x = x$	$x \cdot x = x$
Elemento nulo	$x + 1 = 1$	$x \cdot 0 = 0$
Absorción	$x + x \cdot y = x$	$x \cdot (x + y) = x$
Morgan	$\overline{(x + y)} = \overline{x} \cdot \overline{y}$	$\overline{(x \cdot y)} = \overline{x} + \overline{y}$

3.3.3. Principio de dualidad

Es un metateorema, ya que es un teoremas sobre teoremas.

Si una igualdad o teorema es verdadero también lo será su dual: aquel en el que cambiemos los productos por sumas, y viceversa; y los ceros por unos, y viceversa.

$$\text{si } f_1 = f_2 \text{ entonces } f_1^{D} = f_2^{D} \qquad (3.1)$$

Si observáramos los teoremas anteriores veríamos que en realidad cada uno de ellos son dos, duales entre sí.

La dualidad es extensible a cualquier igualdad, pero en absoluto quiere decir que una función sea igual a su dual $f \neq f^{D}$, por ejemplo $x \cdot y + z \neq (x + y) \cdot z$. Algunas funciones, sin embargo, sí son iguales a su dual, y pasan a denominarse "*self-duals*".

Hay que significar que aunque la aplicación del teorema de Morgan se parece a la aplicación de la dualidad, en absoluto son la misma cosa. En cualquier caso están relacionadas de la siguiente forma: negar una expresión booleana f es idéntico a obtener el dual de la expresión f con las variables negadas.

$$\overline{f(\underline{X})} = f^{D}(\overline{\underline{X}}) \qquad (3.2)$$

Ejemplo 3-1

Comprobar la relación 3.2 para $f=xy+z$.

$$\overline{f(\underline{X})} = \overline{x \cdot y + z} = \overline{x \cdot y} \cdot \overline{z} = (\overline{x} + \overline{y}) \cdot \overline{z}$$

$$f(\overline{\underline{X}}) = \overline{x} \cdot \overline{y} + \overline{z}$$

$$f^{D}(\overline{\underline{X}}) = (\overline{x} + \overline{y}) \cdot \overline{z}$$

$$\overline{f(\underline{X})} = f^{D}(\overline{\underline{X}})$$

Por tanto, negar una expresión booleana es lo mismo que obtener el dual de la expresión con las variables negadas.

3.3.4. Funciones no básicas

Las funciones básicas definidas por Boole son el producto, la suma y el complemento, pero además existen otras funciones que son una combinación de las anteriores. Estas nuevas funciones no son básicas, pero su utilidad, como se verá más adelante, las hace tan importantes o más que las básicas.

Dos de ellas se entienden desde el teorema de Morgan (NAND y NOR) y la tercera implementa la función lógica o exclusiva (XOR). Todas ellas quedan descritas por sus expresiones booleanas y por sus tablas de verdad en la tabla 3-4.

Tabla 3-4
Funciones no básicas.

x	y	$f=\overline{x+y}$	$f=\overline{x\cdot y}$	$f=(x\oplus y)$	$f=\overline{(x\oplus y)}$
0	0	1	1	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	0	0	0	1

Función NOR

Es la negación de una suma (función Not OR).

La suma negada de variables da uno si todas las variables valen cero, en caso contrario el resultado es cero.

$$f=\overline{x+y} \qquad (3.3)$$

Función NAND

Es la negación de un producto (función Not AND).

El producto negado de variables da cero si todas las variables valen uno, en caso contrario el resultado es uno.

$$f=\overline{x\cdot y} \qquad (3.4)$$

Tanto la función NAND como la NOR no cumplen la propiedad asociativa.

$$\overline{\overline{x+y}+z} \neq \overline{x+\overline{y+z}} \qquad (3.5)$$

Función XOR y XNOR

XOR es la función *o exclusiva* (función eXclusive OR).

La suma exclusiva de dos variables es uno si ambas variables son distintas, si son iguales su resultado es cero.

$$f=(x\oplus y)=\overline{x}\cdot y+x\cdot\overline{y} \qquad (3.6)$$

Esta suma tiene un símbolo particular: ⊕ (NAND y NOR también lo tienen, pero no se usa).

La función XNOR es la XOR negada:

$$f=\overline{(x\oplus y)}=x\cdot y+\overline{x}\cdot\overline{y} \qquad (3.7)$$

3.4. Formas normales de una función booleana

Una función tiene un gran número de expresiones equivalentes entre sí, una de ellas es la forma normal o forma canónica.

Esta forma normal puede ser de dos tipos:

- Forma normal disyuntiva. Suma de minitérminos.
- Forma normal conjuntiva. Producto de maxitérminos.

Toda función booleana tiene asociada una única forma normal de cada tipo, con lo que la relación entre una función y su forma normal es biunívoca.

3.4.1. Forma normal disyuntiva. Suma de minitérminos

Antes de describir la forma normal disyuntiva hay que definir el concepto de minitérmino.

Un minitérmino es un producto que contiene todas las variables, negadas o no. Por ejemplo, para tres variables, $x \cdot y \cdot z$, $x \cdot \overline{y} \cdot z$ y $\overline{x} \cdot \overline{y} \cdot z$ son minitérminos, mientras que $x \cdot y$, $x \cdot y + z$, $x \cdot \overline{y \cdot z}$ no lo son.

El valor de cada minitérmino será 1 para una sola combinación de valores de las variables, mientras que para el resto será 0. Según esto podemos establecer una importante y sencilla correspondencia biunívoca entre las filas de la tabla de verdad y el conjunto de minitérminos, asociando a cada fila aquel minitérmino que valga uno para dicha combinación de valores de las variables de entrada.

La tabla 3-5 establece la correspondencia entre minitérminos y filas de la tabla de verdad para tres variables.

Tabla 3-5
Minitérminos y tabla de verdad.

	xyz	Minitérmino	
0	000	$\overline{x}\,\overline{y}\,\overline{z}$	m0
1	001	$\overline{x}\,\overline{y}\,z$	m1
2	010	$\overline{x}\,y\,\overline{z}$	m2
3	011	$\overline{x}\,y\,z$	m3
4	100	$x\,\overline{y}\,\overline{z}$	m4
5	101	$x\,\overline{y}\,z$	m5
6	110	$x\,y\,\overline{z}$	m6
7	111	$x\,y\,z$	m7

De la relación establecida cabe destacar dos aspectos:

- A cada fila de la tabla de verdad formada por ceros y unos se le asigna una expresión booleana.
- La relación establecida es fija y totalmente independiente de la salida -columna resultado-; sólo depende del número de variables de la entrada.

De esta relación se obtiene la forma normal disyuntiva como sigue:

- Plantear la correspondencia entre las filas de la entrada y los minitérminos.
- Plantear la columna resultado de la función.
- Sumar los minitérminos asociados a las filas de la entrada cuyo valor en la columna resultado sea uno.

Resumiendo, sumar los minitérminos asociados a las filas de entrada cuyo valor en la columna salida sea uno. Cuando se obtiene la suma de minitérminos se dice que leemos la tabla de verdad desde los 1.

Este sencillo proceso tiene una gran importancia práctica, pues permite la obtención sistemática de una expresión booleana (forma normal) de cualquier función a partir de su tabla de verdad. Recordemos que la tabla de verdad es una representación basada en los casos particulares, mientras que la expresión booleana es una representación de tipo abstracto; y que la primera facilita la comprensión del problema por nuestra parte, mientras la segunda posibilita el análisis y diseño lógico mediante técnicas de tipo algebraico.

Ejemplo 3-2

Obtener la forma normal disyuntiva de la tabla de verdad.

	x y z	Minitérmino		f
0	0 0 0	$\bar{x}\,\bar{y}\,\bar{z}$	m0	0
1	0 0 1	$\bar{x}\,\bar{y}\,z$	m1	1
2	0 1 0	$\bar{x}\,y\,\bar{z}$	m2	0
3	0 1 1	$\bar{x}\,y\,z$	m3	1
4	1 0 0	$x\,\bar{y}\,\bar{z}$	m4	0
5	1 0 1	$x\,\bar{y}\,z$	m5	0
6	1 1 0	$x\,y\,\bar{z}$	m6	0
7	1 1 1	$x\,y\,z$	m7	1

$$f = m_1 + m_3 + m_7 = \bar{x}\,\bar{y}\,z + \bar{x}\,y\,z + x\,y\,z$$

3.4.2. Forma normal conjuntiva. Producto de maxitérminos

De forma dual a lo expresado para los minitérminos, un maxitérmino es una suma que contiene todas las variables, negadas o no. Cada maxitérmino toma el valor 0 para una única combinación de valores de las variables, siendo 1 para el resto. Lo anterior permite establecer una correspondencia biunívoca entre las filas de la tabla de verdad y los maxitérminos, asignando a cada fila de la tabla de verdad aquel maxitérmino de valor cero para la combinación de valores de las variables correspondientes a la fila de la tabla de verdad (tabla 3-6).

Para obtener la forma normal conjuntiva hay que multiplicar los maxitérminos asociados a las filas de entrada cuyo valor en la columna salida sea cero. Cuando se obtiene el producto de maxitérminos se dice que leemos la tabla de verdad desde los 0.

Tabla 3-6

Maxitérminos y tabla de verdad.

	xyz	Maxitérmino	
0	000	$x + y + z$	M0
1	001	$x + y + \overline{z}$	M1
2	010	$x + \overline{y} + z$	M2
3	011	$x + \overline{y} + \overline{z}$	M3
4	100	$\overline{x} + y + z$	M4
5	101	$\overline{x} + y + \overline{z}$	M5
6	110	$\overline{x} + \overline{y} + z$	M6
7	111	$\overline{x} + \overline{y} + \overline{z}$	M7

Ejemplo 3-3

Obtener la forma normal conjuntiva de la tabla de verdad.

	xyz	Maxitérmino		f
0	000	$(x + y + z)$	M0	0
1	001	$(x + y + \overline{z})$	M1	1
2	010	$(x + \overline{y} + z)$	M2	0
3	011	$(x + \overline{y} + \overline{z})$	M3	1
4	100	$(\overline{x} + y + z)$	M4	0
5	101	$(\overline{x} + y + \overline{z})$	M5	0
6	110	$(\overline{x} + \overline{y} + z)$	M6	0
7	111	$(\overline{x} + \overline{y} + \overline{z})$	M7	1

$$f = M_0 \cdot M_2 \cdot M_4 \cdot M_5 \cdot M_6 = (x + y + z)\,(x + \overline{y} + z)\,(\overline{x} + y + z) \cdot (\overline{x} + y + \overline{z})(\overline{x} + \overline{y} + z)$$

Forma normal conjuntiva de una tabla de verdad

3.4.3. Relación entre las formas normales

Como hemos visto, el proceso para obtener ambas formas normales es dual entre sí, lo que no quiere decir que ambas expresiones sean duales.

Por una parte, minitérmino y maxitérmino son contrarios

$$\overline{m_i} = M_i \quad y \quad \overline{M_i} = m_i \qquad (3.8)$$

Pero con un mayor esfuerzo también vemos que son duales ya que en la obtención de los índices deberíamos cambiar los ceros por unos, y viceversa, así el dual de m_0 sería M_7.

$$m_i^D = M_{7-i} \quad y \quad M_i^D = m_{7-i} \qquad (3.9)$$

Es decir, la clave estaría en cómo designar los minitérminos y maxitérminos.

En cuanto a los procesos de obtención pensemos en una función f, leída desde los unos y desde los ceros. Por ejemplo

$$f_m = m_0 + m_2 + m_4 + m_5 + m_7$$
$$f_M = M_1 \cdot M_3 \cdot M_6$$

Si leyéramos desde los ceros con minitérminos obtendríamos $\overline{f}$. Así

$$\overline{f_m} = m_1 + m_3 + m_6 \text{, y por tanto}$$
$$f_m = \overline{\overline{f_m}} = \overline{m_1 + m_3 + m_6} = \overline{m_1} \cdot \overline{m_3} \cdot \overline{m_6} = M_1 M_3 M_6 = f_M$$
$$f_M = \overline{\overline{f_m}} = f_m \quad \text{y} \quad f_m = \overline{\overline{f_M}} = f_M \qquad (3.10)$$

En ningún caso diremos que $f_m = f_M$[1], o viceversa -aunque en algunos casos pueda serlo-, sino que $f_m = f_M$ por extraño que pueda parecer. Es decir, si multiplicáramos los maxitérminos entre sí aplicando la propiedad distributiva, y redujéramos la expresión aplicando tantos teoremas como haga falta, llegaríamos a la suma de minitérminos, y viceversa desde los minitérminos hacia los maxitérminos.

Ejemplo 3-4

Obtener la forma normal disyuntiva a partir de la conjuntiva.

	x y	Maxitérmino		Minitérmino		f
0	0 0	$(x + y)$	M0	$\overline{x}\,\overline{y}$	m0	0
1	0 1	$(x + \overline{y})$	M1	$\overline{x}\,y$	m1	0
2	1 0	$(\overline{x} + y)$	M2	$x\,\overline{y}$	m2	0
3	1 1	$(\overline{x} + \overline{y})$	M3	$x\,y$	m3	1

$f_m = m_3$

$f_M = M_0 \cdot M_1 \cdot M_2 = (x + y)(x + \overline{y})(\overline{x} + y) = (x \cdot x + x \cdot \overline{y} + y \cdot x + y \cdot \overline{y})(\overline{x} + y) =$

$= (x + x \cdot \overline{y} + y \cdot x)(\overline{x} + y) = (x \cdot \overline{x} + x \cdot y + x \cdot \overline{y} \cdot \overline{x} + x \cdot \overline{y} \cdot y + y \cdot x \cdot \overline{x} + y \cdot x \cdot y) =$

$= x \cdot y + x \cdot y + x \cdot y = x \cdot y = m_3 = f_m$,   es decir, $f_M = f_m$

Hemos visto cómo la forma normal disyuntiva es equivalente a la conjuntiva.

Como siempre podremos pasar de una forma normal a otra, y como ambas son únicas para una función, podemos decir que dos funciones son equivalentes si sus formas normales son idénticas.

Notación compacta

Es muy común utilizar para representar las formas normales una notación compacta, y así evitar lo incómodas que resultan las originales. Esta notación se basa en los símbolos sumatorio y factorial, así:

$$f_m = \sum^{n=3} = (0,2,4,5,7) = m_0 + m_2 + m_4 + m_5 + m_7 = \overline{x}\,\overline{y}\,\overline{z} + \overline{x}y\overline{z} + x\overline{y}\,\overline{z} + x\overline{y}z + xyz$$

$$f_M = \prod^{n=3} = (1,3,6) = M_1 \cdot M_3 \cdot M_6 = (x + y + \overline{z})(x + \overline{y} + \overline{z})(\overline{x} + \overline{y} + z)$$

3.4.4. Otras formas de representación típicas

Las formas normales no son las más usuales en electrónica digital; son útiles para obtener la expresión booleana de una función a partir de su tabla de verdad, pero una vez que se dispone de ella, ésta suele ser manipulada.

El principal objetivo de esta manipulación es conseguir reducir el tamaño de la expresión, para minimizar su coste de implementación. Para ello se intenta eliminar algún minitérmino/maxitérmino, o al menos eliminar alguno de los literales que conforma cada minitérmino/maxitérmino. Este proceso de simplificación es diverso y le dedicaremos el último punto de este capítulo.

La simplificación hace que la expresión booleana deje de ser una forma normal, ahora bien la expresión seguirá siendo una suma de productos -ya no minitérminos- o un producto de sumas -ya no maxitérminos-, conocidos como SOP (Sum Of Products) y POS (Product Of Sums), respectivamente.

La implementación de expresiones en la forma SOP y POS utiliza únicamente puertas AND, OR y NOT.

El uso de expresiones SOP y POS nos lleva a plantearnos cómo pasar de éstas a sus correspondientes formas normales. Para ello hay que multiplicar cada término por $(x + \overline{x})$, donde x es la variable no presente.

Ejemplo 3-5

Obtener la forma normal disyuntiva de la siguiente suma de productos.

$$f = xy+x\overline{z} = xy(z + \overline{z}) + x\overline{z}(y + \overline{y}) = xyz + xy\overline{z} + xy\overline{z}+x\overline{y}\overline{z} =$$
$$= xyz + xy\overline{z} + x\overline{y}\overline{z} = m_7 + m_6 + m_4 = \sum (4,6,7)$$

Forma normal disyuntiva de la expresión f.

Además, una expresión booleana puede ser de cualquier tipo, en este caso si se la quiere convertir en alguna de las anteriores hay que aplicar las reglas de manipulación booleana y los teoremas correspondientes:

Ejemplo 3-6

Convertir en una expresión SOP o POS la siguiente expresión booleana.

$$f = \overline{AB + \overline{\overline{\overline{AD}\,(A + \overline{D})}}} = \overline{AB} \cdot \overline{\overline{\overline{\overline{AD}\,(A + \overline{D})}}} = \overline{AB} \cdot \overline{AD} \cdot (A + \overline{D}) = (\overline{A} + \overline{B}) \cdot (\overline{A} + \overline{D}) \cdot (A + \overline{D})$$

Expresión booleana POS de la función f.

Dentro de este capítulo mostraremos cómo implementar cualquier expresión SOP o POS sólo con puertas NAD o NOR.

3.4.5. Teorema de Shannon

Intuitivamente, la obtención de la forma normal disyuntiva puede parecer clara: sumar los minitérminos relacionados con 1 en la salida. Sin embargo, este proceso

intuitivo necesita de una justificación teórica, ya que al fin y al cabo éste es el objetivo del álgebra de Boole: formalizar algebraicamente el pensamiento lógico.

El teorema de Shannon cumple este propósito. Mostraremos para tres variables la aplicación del teorema de Shannon para obtener la forma normal.

Según el teorema de Shannon, para toda variable x, y, z perteneciente al álgebra de Boole, se cumple:

$$f(x,y,z) = x \cdot f(1,y,z) + \overline{x} \cdot f(0,y,z) \qquad (3.11)$$

Si aplicamos sucesivamente el teorema de Shannon para las variables z e y.

$$f(x,y,z) = x \cdot f(1,y,z) + \overline{x} \cdot f(0,y,z) = x \cdot [(y \cdot f(1,1,z) + \overline{y} \cdot f(1,0,z))] + \overline{x} \cdot [(y \cdot f(0,1,z) + \overline{y} \cdot f(0,0,z))] = x \cdot [y \cdot (z \cdot f(1,1,1) + \overline{z} \cdot f(1,1,0)) + \overline{y} \cdot (z \cdot f(1,0,1) + \overline{z} \cdot f(1,0,0))] + \overline{x} \cdot [y \cdot (z \cdot f(0,1,1) + \overline{z} \cdot f(0,1,0)) + \overline{y} \cdot (z \cdot f(0,0,1) + \overline{z} \cdot f(0,0,0))] =$$

$$= x\,y\,z \cdot f(1,1,1) + x\,y\,\overline{z} \cdot f(1,1,0) + x\,\overline{y}\,z \cdot f(1,0,1) + x\,\overline{y}\,\overline{z} \cdot f(1,0,0) + \overline{x}\,y\,z \cdot f(0,1,1) + \overline{x}\,y\,\overline{z} \cdot f(0,1,0) + \overline{x}\,\overline{y}\,z \cdot f(0,0,1) + \overline{x}\,\overline{y}\,\overline{z} \cdot f(0,0,0)$$

Es decir:

$$f(x,y,z) = x\,y\,z \cdot f(1,1,1) + x\,y\,\overline{z} \cdot f(1,1,0) + x\,\overline{y}\,z \cdot f(1,0,1) + x\,\overline{y}\,\overline{z} \cdot f(1,0,0) + \overline{x}\,y\,z \cdot f(0,1,1) + \overline{x}\,y\,\overline{z} \cdot f(0,1,0) + \overline{x}\,\overline{y}\,z \cdot f(0,0,1) + \overline{x}\,\overline{y}\,\overline{z} \cdot f(0,0,0) \qquad (3.12)$$

Donde, por ejemplo f (1,1,1) es el valor que toma la función f para x = 1, y = 1 y z = 1.

Ejemplo 3-7

Obtener la forma normal disyuntiva correspondiente a la tabla de verdad adjunta.

	x y z	Minitérmino		f
0	0 0 0	$\overline{x} \cdot \overline{y} \cdot \overline{z}$	m0	0
1	0 0 1	$\overline{x} \cdot \overline{y} \cdot z$	m1	1
2	0 1 0	$\overline{x} \cdot y \cdot \overline{z}$	m2	0
3	0 1 1	$\overline{x} \cdot y \cdot z$	m3	1
4	1 0 0	$x \cdot \overline{y} \cdot \overline{z}$	m4	0
5	1 0 1	$x \cdot \overline{y} \cdot z$	m5	0
6	1 1 0	$x \cdot y \cdot \overline{z}$	m6	0
7	1 1 1	$x \cdot y \cdot z$	m7	1

Aplicando el teorema de Shannon:

$$f(x,y,z) = x\,y\,z \cdot f(1,1,1) + x\,y\,\overline{z} \cdot f(1,1,0) + x\,\overline{y}\,z \cdot f(1,0,1) + x\,\overline{y}\,\overline{z} \cdot f(1,0,0) + \overline{x}\,y\,z \cdot f(0,1,1) + \overline{x}\,y\,\overline{z} \cdot f(0,1,0) + \overline{x}\,\overline{y}\,z \cdot f(0,0,1) + \overline{x}\,\overline{y}\,\overline{z} \cdot f(0,0,0)$$

Sustituyendo la función f por sus valores:

$$f(x,y,z) = x\,y\,z \cdot 1 + x\,y\,\overline{z} \cdot 0 + x\,\overline{y}\,z \cdot 0 + x\,\overline{y}\,\overline{z} \cdot 0 + \overline{x}\,y\,z \cdot 1 + \overline{x}\,y\,\overline{z} \cdot 0 + \overline{x}\,\overline{y}\,z \cdot 1 + \overline{x}\,\overline{y}\,\overline{z} \cdot 0$$

$$f(x,y,z) = x\,y\,z + \overline{x}\,y\,z + \overline{x}\,\overline{y}\,z = m_7 + m_3 + m_1$$

Obtención de la forma normal aplicando el teorema de Shannon.

El método aquí presentado es válido para los minitérminos; el correspondiente a los maxitérminos es el dual de éste.

El procedimiento seguido en el ejemplo es muy farragoso y no es el habitual; en general, para obtener f se suman directamente los minitérminos, o se multiplican los maxitérminos.

3.5. Implementación de funciones booleanas

Además de las bases técnicas aportadas por Boole, Morgan, Huntington, Shannon, etc., el importante desarrollo de la electrónica digital, y junto a ella del álgebra de Boole, se ha debido a la capacidad tecnológica del hombre para construir dispositivos eléctricos capaces de operar como operadores lógicos.

La implementación tecnológica de las distintas puertas ha pasado por muy diversas etapas en muy corto tiempo, pero los tipos de puertas básicas implementados siempre han sido los mismos. La tabla 3-7 recoge la función que implementan y el símbolo típico.

Tabla 3-7
Operaciones lógicas y puertas.

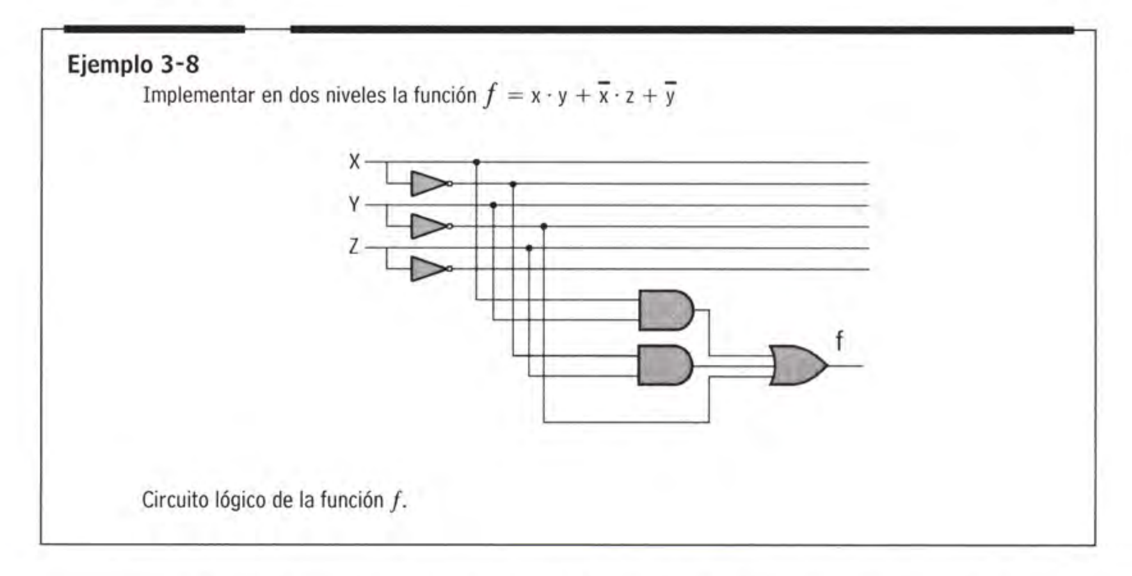
Operación lógica	Puerta lógica
NOT	X → $\overline{X}$
AND	X, Y → X·Y
OR	X, Y → X+Y
XOR	X, Y → X⊕Y
NAND	X, Y → $\overline{X \cdot Y}$
NOR	X, Y → $\overline{X+Y}$

3.5.1. Implementación de funciones POS y SOP

La implementación más popular de funciones SOP o POS es la implementación en dos niveles. Si la función a implementar está en forma de suma de productos, el primer nivel contiene tantas puertas AND como productos, mientras el segundo nivel tiene una única puerta OR, que recibe como entrada las salidas de las puertas AND y cuya salida es la propia función. Esta red se denomina red AND-OR.

Ejemplo 3-8

Implementar en dos niveles la función $f = x \cdot y + \overline{x} \cdot z + \overline{y}$



Circuito lógico de la función *f*.

A los dos niveles antes indicados hay que añadir un posible nivel de puertas NOT encargadas de preparar las variables de entrada para negarlas. Pero también cabe preguntarse por qué si existe x no va a existir $\overline{x}$; así, muchas veces el circuito contempla x y $\overline{x}$, lo que se denomina en inglés "*doubled-rail*".

En el anterior circuito vemos que $\overline{y}$ no pasa por ninguna puerta AND ya que este producto consta de un solo término. En algunas implementaciones deben aparecer todas los niveles, y así $\overline{y} = \overline{y} \cdot \overline{y}$, resultando el circuito de la figura 3-1.

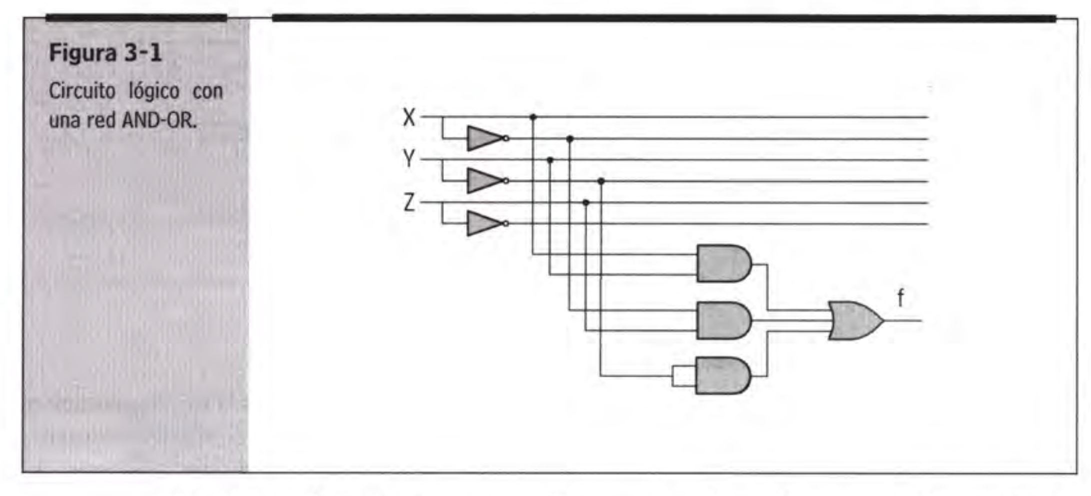


Figura 3-1

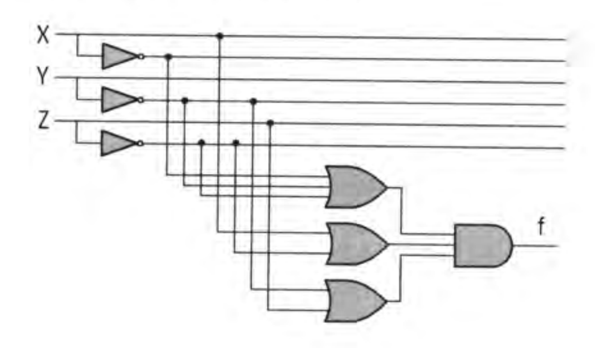
Circuito lógico con una red AND-OR.

En el caso de funciones expresadas como producto de sumas la disposición de niveles es la dual, en el primero puertas OR y en el segundo una puerta AND. Esta red se denomina red OR-AND.



Ejemplo 3-9

Implementar en dos niveles $f = (\bar{x} + \bar{y} + \bar{z})(x + \bar{z})(\bar{y} + z)$



Circuito lógico de la función f.

3.5.2. Implementación de funciones con puertas NAND y NOR

Cualquier función booleana puede ser implementada sólo con puertas NAND o sólo con puertas NOR. Esto supone una gran ventaja ya que las puertas lógicas se distribuyen agrupadas en circuitos integrados. Así, para una implementación AND/OR harían falta dos tipos de circuito integrado y sus correspondientes alimentaciones, mientras que con NAND/NOR bastaría con un solo circuito integrado. Además, tecnológicamente es más fácil fabricar una puerta NAND que una AND: podríamos decir que una AND es una NAND negada.

Para que lo anterior sea aplicable bastará con que las tres operaciones básicas, NOT, AND y OR, sean implementables sólo con NAND o sólo con NOR. Para establecer esta relación utilizaremos como soporte teórico el teorema de Morgan.

3.5.2.1. Funciones básicas con NAND/NOR

El objetivo es obtener las expresiones booleanas y sus correspondientes circuitos lógicos que implementen cada función básica sólo con NAND o sólo con NOR, aunque para ello haya que utilizar un mayor número de puertas. El resumen de lo obtenido aparece en la tabla 3-8.

Función negación NOT $\quad f = \bar{x}$

Con puertas NOR: $\quad f = \overline{x + x}$

Con puertas NAND: $\quad f = \overline{x \cdot x}$

Función suma OR $\quad f = x + y$

Con puertas NOR: $\quad f = \overline{\overline{x + y}}$

Con puertas NAND: $\quad f = \overline{\bar{x} \cdot \bar{y}}$

Función producto AND	$\mathbf{f = x \cdot y}$
Con puertas NOR:	$f = \overline{\overline{x} + \overline{y}}$
Con puertas NAND:	$f = \overline{\overline{x \cdot y}}$

Tabla 3-8

Operaciones básicas con puertas NAND/NOR.

Operación	Con NOR	Con NAND
$f = \overline{x}$	$f = \overline{x + x}$	$f = \overline{x \cdot x}$
X → $\overline{X}$	$\overline{X+X}=\overline{X}$	$\overline{X \bullet X}=\overline{X}$
$f = x+y$	$f = \overline{\overline{x+y}}$	$f = \overline{\overline{x} \cdot \overline{y}}$
X, Y → $X+Y$	X, Y → $\overline{X+Y}$ → $\overline{\overline{X+Y}}=X+Y$	$\overline{X}$, $\overline{Y}$ → $\overline{\overline{X} \cdot \overline{Y}}=X+Y$
$f = x \cdot y$	$f = \overline{\overline{x}+\overline{y}}$	$f = \overline{\overline{x \cdot y}}$
X, y → $X \bullet Y$	$\overline{X}$, $\overline{Y}$ → $\overline{\overline{X}+\overline{Y}}=X \cdot Y$	X, Y → $\overline{X \bullet Y}$ → $\overline{\overline{X \bullet Y}}=X \bullet Y$

Vemos que el comportamiento entre NAND y NOR es el dual. Además vemos cómo la implementación del producto con NAND y de la suma con NOR son un poco torpes, aunque no falsas. Recordemos que el objetivo es implementar todas las funciones básicas sólo con NAND o sólo con NOR, sin importar cómo.

Si toda función básica puede ser implementada con sólo NAND o NOR, podemos asegurar que cualquier función se puede implementar sólo con NAND o sólo con NOR, con lo que mejoraríamos la calidad de la implementación aunque perdiéramos claridad en la representación.

3.5.2.2. Implementación de expresiones SOP y POS con puertas NAND/NOR

Ambas expresiones son muy sencillas de implementar con NAND/NOR, como veremos a continuación. Simplemente hay que manipular las expresiones, aplicando el teorema de Morgan, hasta que únicamente aparezcan las operaciones NAND o NOR.

Expresiones SOP. Suma de productos

1. Con puertas NAND.

Los pasos son:

- Negar dos veces la función SOP.
- Aplicar Morgan al conjunto.
- Cada sumando queda negado y el conjunto también.
- Repasar el resultado aplicando de nuevo el teorema de Morgan.

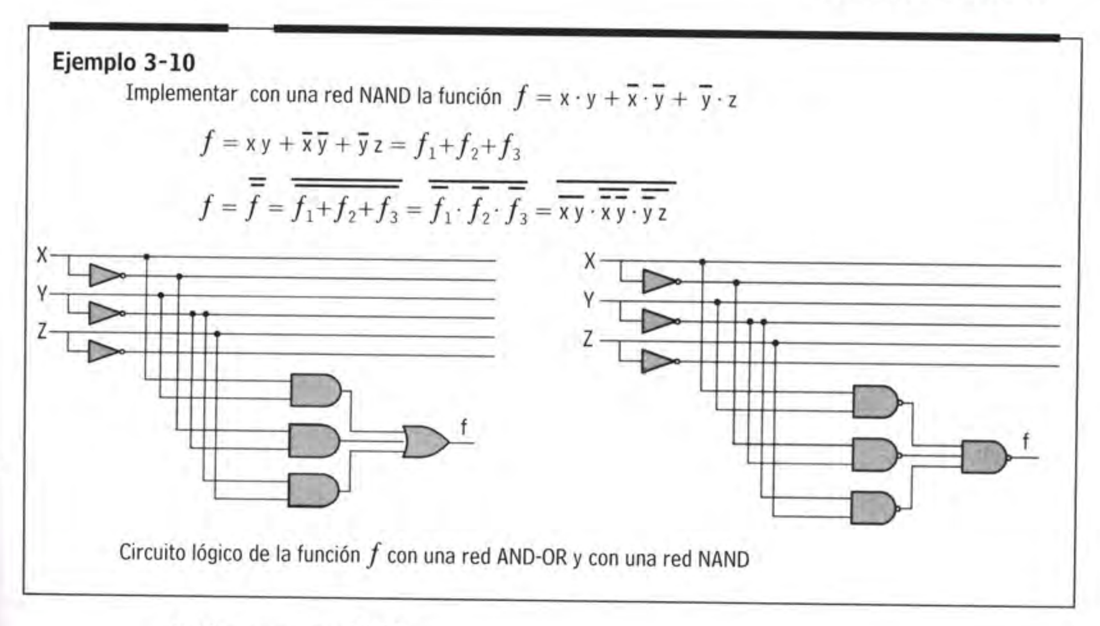
La expresión a implementar es la original con los sumandos negados, la función negada, los + sustituidos por · y las variables sin negar.

Ejemplo 3-10

Implementar con una red NAND la función $f = x \cdot y + \bar{x} \cdot \bar{y} + \bar{y} \cdot z$

$$f = x\,y + \bar{x}\,\bar{y} + \bar{y}\,z = f_1 + f_2 + f_3$$

$$f = \overline{\overline{f}} = \overline{\overline{f_1 + f_2 + f_3}} = \overline{\overline{f_1} \cdot \overline{f_2} \cdot \overline{f_3}} = \overline{\overline{x\,y} \cdot \overline{\bar{x}\,\bar{y}} \cdot \overline{\bar{y}\,z}}$$



Circuito lógico de la función f con una red AND-OR y con una red NAND

2. Con puertas NOR.

Los pasos son:

- Negar dos veces la función SOP.
- Aplicar Morgan a cada sumando.
- Cada sumando se convierte en una suma negada con las variables negadas.
- Repasar el resultado aplicando de nuevo el teorema de Morgan.

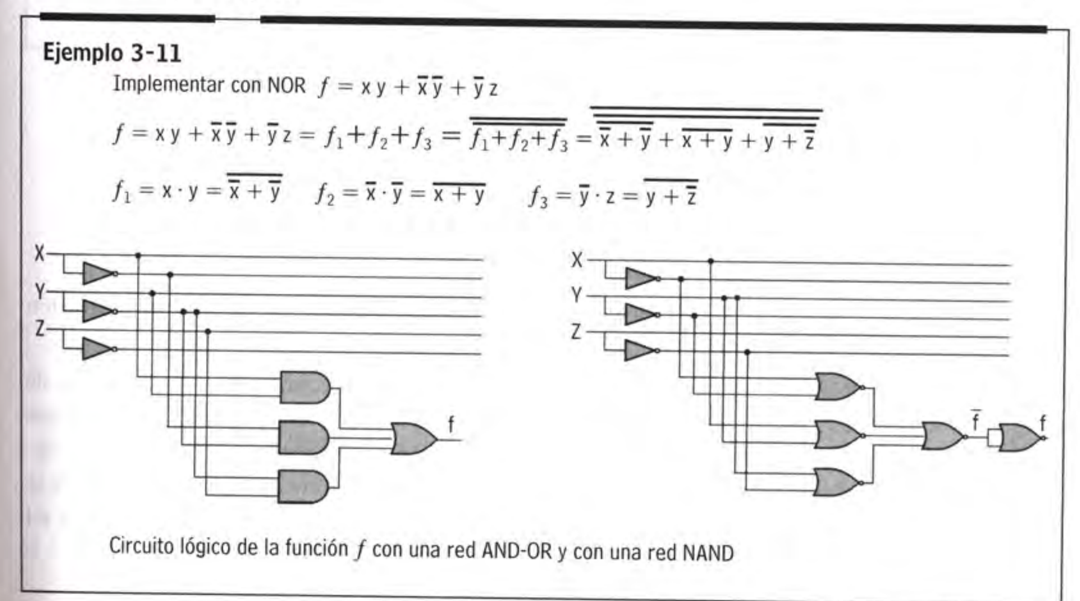
La expresión a implementar es la original con los sumandos convertidos en sumas negadas, la función negada dos veces, los · sustituidos por + y las variables negadas.

Ejemplo 3-11

Implementar con NOR $f = x\,y + \bar{x}\,\bar{y} + \bar{y}\,z$

$$f = x\,y + \bar{x}\,\bar{y} + \bar{y}\,z = f_1 + f_2 + f_3 = \overline{\overline{f_1 + f_2 + f_3}} = \overline{\overline{\overline{\bar{x} + \bar{y}} + \overline{x + y} + \overline{y + \bar{z}}}}$$

$$f_1 = x \cdot y = \overline{\bar{x} + \bar{y}} \qquad f_2 = \bar{x} \cdot \bar{y} = \overline{x + y} \qquad f_3 = \bar{y} \cdot z = \overline{y + \bar{z}}$$



Circuito lógico de la función f con una red AND-OR y con una red NAND

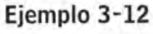
Expresiones POS. Productos de sumas

1. Con puertas NAND.

Los pasos son:

- Negar dos veces la función POS.
- Aplicar Morgan a cada factor.
- Cada factor se convierte en un producto negado con las variables negadas.
- Repasar el resultado aplicando de nuevo el teorema de Morgan.

La expresión a implementar es la original con los sumandos convertidos en factores negados, la función negada dos veces, los + sustituidos por · y las variables negadas.

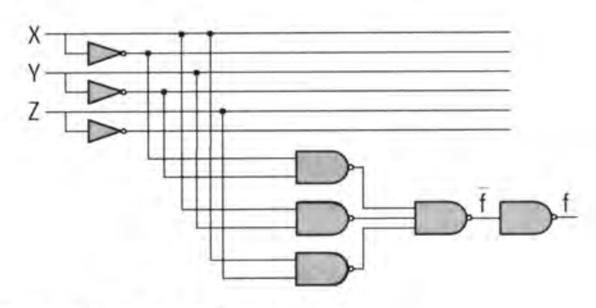


Ejemplo 3-12

Implementar con NAND $f = (x + y)\,(\bar{x} + \bar{y})\,(\bar{x} + \bar{z})$

$$f = (x + y)\,(\bar{x} + \bar{y})\,(\bar{x} + \bar{z}) = f_1 \cdot f_2 \cdot f_3 = \overline{\overline{f_1 \cdot f_2 \cdot f_3}} = \overline{\overline{\overline{\bar{x} \cdot \bar{y}} \cdot \overline{x \cdot y} \cdot \overline{x \cdot z}}}$$

$$f_1 = x + y = \overline{\bar{x} \cdot \bar{y}} \qquad f_2 = \bar{x} + \bar{y} = \overline{x \cdot y} \qquad f_3 = \bar{x} + \bar{z} = \overline{x \cdot z}$$



Circuito lógico de la función f con una red NAND.

2. Con puertas NOR.

Los pasos son:

- Negar dos veces la función POS.
- Aplicar Morgan al conjunto.
- Cada factor queda negado y el conjunto también.
- Repasar el resultado aplicando de nuevo el teorema de Morgan.

La expresión a implementar es la original con los factores negados, la función negada, los · sustituidos por + y las variables sin negar.

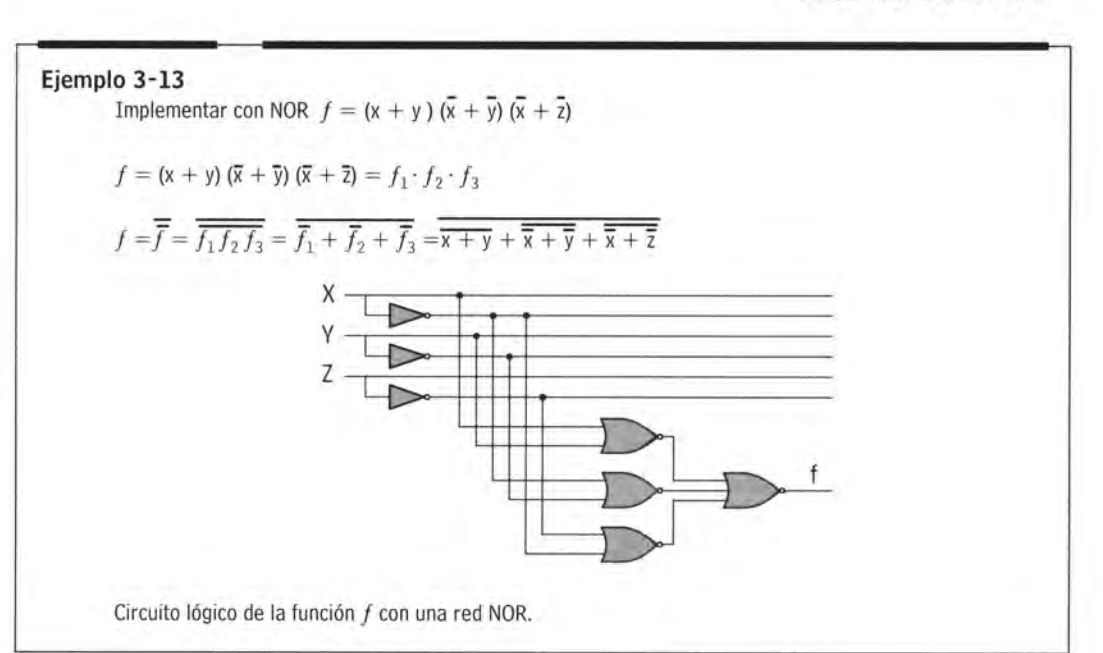
A la hora de implementar con NAND/NOR basta con respetar el teorema de Morgan, tener cuidado con la relación de precedencia al sustituir los productos por sumas y viceversa, y por último prestar especial atención a los términos formados por una sola variable. Finalmente, debemos comprobar la validez de la expresión obtenida aplicando el teorema de Morgan de forma visual, para así evitar despistes.

Ejemplo 3-13

Implementar con NOR $f = (x + y)(\bar{x} + \bar{y})(\bar{x} + \bar{z})$

$$f = (x + y)(\bar{x} + \bar{y})(\bar{x} + \bar{z}) = f_1 \cdot f_2 \cdot f_3$$

$$f = \bar{\bar{f}} = \overline{\overline{f_1 f_2 f_3}} = \overline{\bar{f}_1 + \bar{f}_2 + \bar{f}_3} = \overline{\overline{x + y} + \overline{\bar{x} + \bar{y}} + \overline{\bar{x} + \bar{z}}}$$



Circuito lógico de la función f con una red NOR.

Observando ambas implementaciones cabe concluir que no es lo mismo implementar con NAND o NOR según sea el caso. En el caso de expresiones SOP es preferible implementarlas con puertas NAND, cuyo circuito tiene en general una puerta menos que con NOR. De forma dual podemos decir que las expresiones POS son preferiblemente implementables con puertas NOR.

Si la expresión a implementar no fuera ni POS ni SOP habría que manipularla booleanamente hasta conseguir reducirla a una expresión SOP o POS, o aplicar el teorema de Morgan de forma intuitiva a cada caso en particular.

3.5.3. Otras implementaciones

Actualmente, la tendencia consiste en utilizar PLD's y otros dispositivos para implementar funciones lógicas, quedando las puertas lógicas relegadas a funciones muy específicas. Así, lo explicado en este punto queda más como una realidad teórica y didáctica que como una opción profesional.

3.6. Otras funciones lógicas

Tres son las operaciones básicas +, · y ¯, pero además de éstas pueden plantearse otras, algunas de ellas, como ya hemos visto, con interés para el diseño digital: NAND, NOR, XOR y XNOR. Ahondando, cabe preguntarse cuántas operaciones distintas pueden plantearse entre un número de variables. La respuesta es $2^{(2^n)}$, donde n es el número de variables. Por ejemplo, entre dos variables x e y pueden plantearse 16 operaciones booleanas.

En la tabla 3.9 se muestra el número de operaciones para distintos números de variables. La última columna indica cuáles de las posibles operaciones son realmente distintas, y no un simple cambio entre variables.

Tabla 3-9
Número de funciones booleanas.

n	2n	$2^{(2^n)}$	Nº Operaciones
1	2	4	3
2	4	16	6
3	8	256	22
4	16	65536	402
5	32	4.294.967.296	1.228.1581

En la tabla 3.10 se describen las 16 operaciones posibles entre dos variables.

Si observamos las dieciséis funciones podemos agruparlas como:

- Dos constantes, 0 y 1.
- Cuatro operaciones de una variable: transferencia y complemento.
- Diez operaciones con dos variables, seis de ellas son: AND, OR, NAND, NOR, XOR y XNOR.

Tabla 3-10
Funciones booleanas posibles para dos variables.

x	y	0	$\bar{x}\cdot\bar{y}$	$\bar{x}\cdot y$	$\bar{x}$	$x\cdot\bar{y}$	$\bar{y}$	$x\oplus y$
0	0	0	1	0	1	0	1	0
0	1	0	0	1	1	0	0	1
1	0	0	0	0	0	1	1	1
1	1	0	0	0	0	0	0	0

$\bar{x}+\bar{y}$	$x\cdot y$	$\overline{x\oplus y}$	y	$\bar{x}+y$	x	$x+\bar{y}$	$x+y$	1
1	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	1	1
1	0	0	0	0	1	1	1	1
0	1	1	1	1	1	1	1	1

En principio las 16 operaciones podrían ser implementadas tecnológicamente para su distribución y uso en diseño digital. Es claro que deben ser implementadas las operaciones AND, NOR, NOT, NAND, NOR y XOR, pero podemos plantearnos qué propiedades debe tener una función para ser implementada:

- Principalmente la facilidad y la economía de ella derivadas para fabricar la puerta con elementos tecnológicos.
- La posibilidad de que el número de entradas pueda ser aumentado a más de dos entradas.
- El operador preferiblemente debe cumplir las propiedades asociativa y conmutativa.
- La posibilidad de implementar la puerta lógica con otras ya existentes.

Respecto de las puertas anteriores cabe destacar que ni NAND ni NOR cumplen la propiedad asociativa, pero a cambio son autónomas.

Las puertas lógicas pueden ser implementadas de diversas formas, conformando diversas familias tecnológicas, cada una con sus ventajas y desventajas, como se verá en el capítulo 5.

3.7. Lógica multivaluada

Todos los circuitos planteados utilizan lógica bivaluada, es más, el álgebra de Boole ha sido definida en este capítulo axiomáticamente con variables bivaluadas, pero el álgebra de Boole es definible para variables con más de dos valores posibles, conformando la lógica multivaluada (MVL: MultiValued Logic).

En principio su uso puede parecer absurdo, ya que el razonamiento lógico utiliza premisas con sólo dos valores, verdadero y falso. Las ventajas que plantea la lógica multivaluada son de tipo preferentemente tecnológico:

- Al aumentar la cantidad de información de cada variable disminuirán las conexiones necesarias en un circuito integrado; también se reducirán el número de patitas de los CI's.
- Las variables multivaluadas aumentan la densidad de información, y así, para implementar una función booleana será menor el número de puertas necesarias, reduciéndose el número de circuitos integrados a utilizar.

Las anteriores ventajas quedan compensadas por las siguientes desventajas:

- La lógica bivaluada es la más cercana al razonamiento humano: la utilización de una lógica multivaluada supondría un gran esfuerzo en aprendizaje.
- Los múltiples valores de una variable deben distribuirse dentro del rango de voltajes, por ejemplo 0-5 V; con lo que la tolerancia se hace más crítica y el efecto del ruido puede ser mayor, pudiendo modificarse con facilidad el valor actual de la variable.
- Una solución a lo anterior es aumentar el rango de voltaje y así dejar un mayor rango de voltaje a cada valor de variable, pero esta solución trae consigo una reducción en la velocidad de operación.
- La industria electrónica ha invertido una gran cantidad de dinero en investigar y desarrollar productos digitales binarios. A su vez, un gran número de industrias han desarrollado sus productos pensando en circuitos digitales bivaluados. El cambio de lógica y tecnología supondría un esfuerzo logístico y económico muy grande.

Actualmente nada hace pensar que se vaya a producir un cambio en la lógica multivaluada. De hecho, son casi inexistentes los circuitos multivaluados fabricados; su uso está muy restringido, a memorias ROM por ejemplo, y su potencial lógico no es utilizado.

La posible definición del álgebra de Boole con variables multivaluadas nos lleva a rechazar la definición axiomática dada al comienzo del capítulo, pues ésta presuponía variables bivaluadas.

La siguiente definición contempla la lógica multivaluada -el lector podrá encontrarla en muchos textos como definición de la lógica bivaluada-. Es la más clásica e histórica y se basa en los postulados enunciados por E.V. Huntington en 1904:

Definición del álgebra booleana multivaluada

Un conjunto booleano de variables, cuyo valor pertenece a un número finito de valores, y dos operaciones + y ·, conforman una estructura de álgebra de Boole si:

P.1. Álgebra cerrada

El conjunto booleano de variables es cerrado respecto de + y ·, o sea, la suma o el producto de variables es otra variable perteneciente al álgebra de Boole.

P.2. Elemento identidad

$$x + 0 = x \qquad x \cdot 1 = x$$

P.3. Propiedad conmutativa.

$$x + y = y + x \qquad x \cdot y = y \cdot x$$

P.4. Propiedad distributiva.

$$x \cdot (y + z) = x \cdot y + x \cdot z$$

$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

P.5. Definición de complemento.

$$x + \overline{x} = 1 \qquad x \cdot \overline{x} = 0$$

P.6. Dos variables x e y

Existen al menos dos variables x e y distintas:

$$x \neq y$$

Como el lector puede observar los postulados de Huntington han sido expresados como teoremas básicos en la definición de álgebra de Boole utilizada en el texto.

Aquí abandonamos la lógica multivaluada, pues sus contenidos quedan fuera del nivel de este libro.

3.8. Simplificación de funciones booleanas

Al simplificar una función booleana buscamos reducir su tamaño sin modificarla. De esta manera, la función a implementar tendrá un coste menor.

Por ejemplo, la función $f = A\overline{B} + AB + \overline{A}B$ simplificada queda en $f_s = A+B$. El comportamiento lógico de f y f_s es idéntico, pero f necesita de 3 puertas AND y 1 OR, y f_s de 1 puerta OR.

Así pues, simplificar es reducir el número de términos y de literales de la función original. Al reducir los términos reducimos el número de OR (caso de SOP), y al reducir los literales reducimos el número de AND (caso de SOP). De todos modos el criterio de simplificación no es único, y de hecho éste varía mucho con los avances en microelectrónica. Así, simplificar puede ser:

- reducir únicamente el número de términos,
- reducir el número de CI's,
- evitar riesgos estáticos,
- etc.

Además, los continuos avances en diseño digital asistido por computador liberan al usuario de este proceso, convirtiéndose en una tecla más a pulsar. Sin embargo, es importante que el lector domine la simplificación *a mano* para aplicar con corrección y estilo los métodos computacionales.

Existen multitud de métodos de simplificación, destacando entre ellos:

- Algebraico. Método teórico poco operativo.
- Quine-McCluskey. Método potente de gran valor computacional, pero de laboriosa aplicación a mano.
- Diagramas de Veitch-Karnaugh. Método gráfico poco potente, pero de fácil aplicación manual.
- Métodos computacionales desarrollados para ser implementados únicamente en computadoras: ESPRESSO, McBOOLE, CAMP-DEUSTO, etc.

En nuestro caso sólo abordaremos el planteado por Veitch y Karnaugh en los años 50, tanto por su idoneidad para el nivel de este libro como por su utilidad para sistemas pequeños.

3.9. Método de Veitch-Karnaugh

Este método permite obtener de forma visual y cómoda la expresión mínima de una original. Para aplicar el método, previamente hay que conocer los diagramas de V-K.

3.9.1. Diagramas de Veitch-Karnaugh

A principios de los años 50 Veitch planteó un diagrama que facilitaba la simplificación. Dicho diagrama fue modificado por Karnaugh hasta dejarlo como lo conocemos ahora.

Un diagrama de V-K es un panel en el que las entradas y salidas comparten un mismo espacio, no como en la tabla de verdad con las entradas a la izquierda y las salidas a la derecha. Además, en un V-K las casillas próximas entre sí son adyacentes (distancia 1). Es decir, casillas que comparten un lado sólo se diferencian en un bit.

La superposición de entradas y salidas y la adyacencia de casillas nos permite simplificar *con la vista*.

Si bien pueden plantearse V-K de cualquier número de variables, lo normal es no pasar de seis. A continuación describiremos los V-K de 2, 3, 4, 5 y 6 variables.

3.9.2. Disposición de los V-K de varias variables

La disposición del V-K debe asegurar que cada fila de la entrada de la tabla de verdad tenga asociada una casilla del V-K, y viceversa, estableciéndose una

correspondencia biunívoca entre la tabla de verdad y el diagrama de V-K. Además de este emparejamiento, debe asegurarse que todas las casillas que rodeen a otra sean adyacentes a ésta.

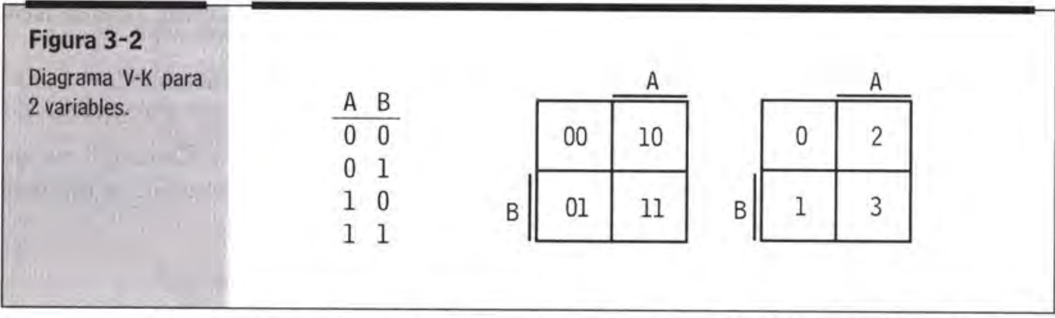
Existen varias formas de plantear el V-K, según cómo se dispongan las variables en el V-K. En nuestro caso las variables booleanas se denominarán A, B, C..., siendo la A la de más peso.

Como se verá en los siguientes apartados, la disposición de las variables en el V-K sigue en cierta forma el código Gray, lo que asegura la adyacencia entre casillas.

3.9.2.1. Diagrama de V-K de 2 variables

En este caso el V-K debe tener 4 casillas, como se puede ver en la figura 3-2.

Figura 3-2
Diagrama V-K para 2 variables.



Vemos cómo cada casilla se corresponde con una fila de la tabla de verdad.

Las casillas debajo de la raya tienen un valor 1 para la variable señalada, y un cero en caso contrario. Siguiendo esta regla hemos completado todo el V-K.

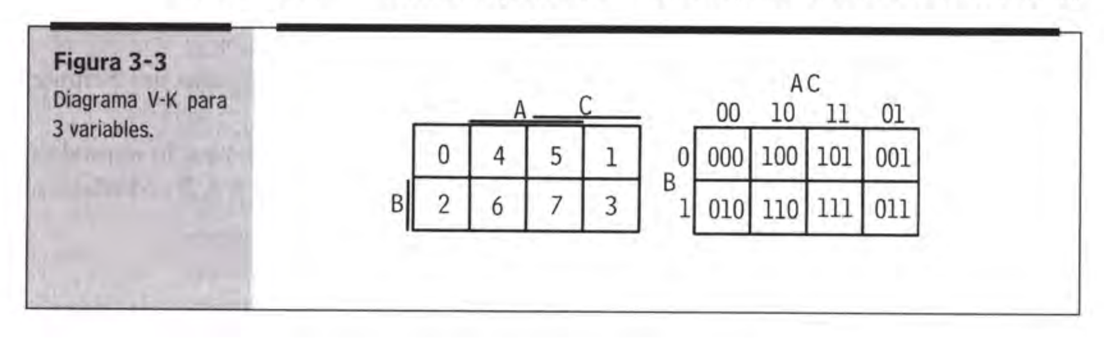
Para distinguir qué fila es cada casilla podemos optar por la descripción explícita con bits (a la izquierda), o por la descripción implícita con números decimales (derecha). La más utilizada es esta última, aunque lo más normal es no usar ninguna de ellas, recordando la distribución de memoria.

3.9.2.2. Diagrama de V-K de 3 variables

En la figura 3-3 vemos cómo quedan codificadas las casillas.

Vemos que a la derecha, izquierda, arriba o abajo de cada casilla se encuentra

Figura 3-3
Diagrama V-K para 3 variables.



otra adyacente, de forma que casillas que visualmente están próximas son adyacentes. Pero también están adyacentes las casillas 2 y 3, y las 1 y 0, como si el V-K fuera un cilindro.

3.9.2.3. Diagrama de V-K de 4 variables

En la figura 3-4 vemos cómo quedan dispuestas las casillas según el orden y peso elegidos para las variables. Queda para el lector el comprobar que dicha distribución es acertada.

Si el lector se fija un poco observará que aunque las casillas parecen desordenadas, realmente no lo están tanto: arriba, abajo, derecha e izquierda. Antes o después habrá que memorizar la posición de las casillas, para no tener que obtenerlas cada vez.

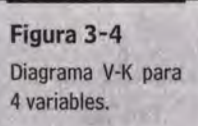
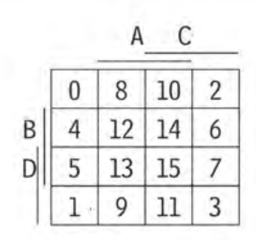


Figura 3-4
Diagrama V-K para 4 variables.



3.9.2.4. Diagramas de V-K de 5 y 6 variables

Por su tamaño e incomodidad no suelen ser usados (sobre todo el de 6 variables), siendo su disposición la de la figura 3-5.

Lo más normal no es plantear un V-K de 32 o 64 casillas, sino dos de 16 o cuatro de 16. De esta forma la simplificación es más larga, pero también más sencilla.

Figura 3-5
Diagrama V-K para 5 variables.

A=0 (B, D; C, E)

0	8	10	2
4	12	14	6
5	13	15	7
1	9	11	3

A=1 (B, D; C, E)

16	24	26	18
20	28	30	22
21	29	31	23
17	25	27	19

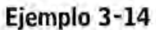
En el primer V-K, explícitamente A = 0, siendo A = 1 en el siguiente.

La disposición de un V-K de 6 variables queda para el lector, recordándole que debe plantear cuatro V-K de 4 variables: AB = 0, AB = 01, AB = 10 y AB = 11.

3.9.3. Obtención del diagrama de V-K de una función booleana

Se puede obtener el V-K de cualquier función booleana, sea cual sea la representación de ésta. Ahora bien, en un proceso de diseño lo más normal es obtener el o los V-K correspondientes a una tabla de verdad.

Si partimos de una tabla de verdad, simplemente debemos recordar que en el punto anterior hemos relacionado cada fila con una casilla. Sólo resta escribir sobre cada casilla el correspondiente valor de la salida.



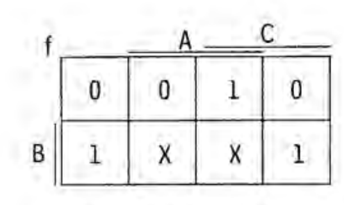
Ejemplo 3-14

Obtener el V-K de las dos funciones booleanas adjuntas.

A	B	f
0	0	0
0	1	1
1	0	1
1	1	1

f		A
	0	1
B	1	1

$f = \Sigma\,(1,2,3)$



f		A	A, C	C
	0	0	1	0
B	1	X	X	1

$f = \Sigma\,(2,3,5) + (6,7)$

En ambos ejemplos hemos escrito tanto los 0, como los 1, como las X. Esto no es lo normal, lo normal es escribir sólo los 0 o los 1, además de las X.

Ejemplo 3-15

Obtener los V-K de las siguientes funciones booleanas.

$f1 = \sum^{4} (0,2,3,4,9,11,12,13,14)$

$f2 = \sum^{4} (4,7,9,12\text{-}15)$

$f3 = \sum^{4} (0\text{-}5,7,8,10\text{-}15)$

$f4 = \sum^{4} (2\text{-}5,10\text{-}13) + X = \sum^{4} (7,15)$

f1

		A	A, C	C
	1			1
B	1	1	1	
B, D		1		
D		1	1	1

f2

		A	A, C	C
B	1	1	1	
B, D		1	1	1
D		1		

f3

		A	A, C	C
	1	1	1	1
B	1	1	1	
B, D	1	1	1	1
D	1		1	1

f4

		A	A, C	C
			1	1
B	1	1		
B, D	1	1	X	X
D			1	1

Ahora que ya sabemos plantear y completar un V-K, estamos en condiciones de aprender a simplificar.

3.9.4 Simplificación mediante V-K

Reescribamos desde los V-K la definición de simplificar: simplificar es rodear todos los 1 (0) con el menor número posible de lazos del mayor tamaño posible. Queda saber qué es un lazo y cómo se forma.

Un lazo es una agrupación de 2^n casillas adyacentes entre sí. Resulta que como las casillas adyacentes son próximas, podemos decir que un lazo es una agrupación de casillas próximas formando:

- cuadrados,
- filas,
- columnas,
- etc.

Para lo anterior hay que tener en cuenta que en un V-K:

- las partes superior e inferior están próximas entre sí,
- la partes derecha e izquierda están próximas entre sí,
- para un V-K de 4 y superiores, las cuatro esquinas están próximas entre sí.

Lo anterior supone que el V-K, además de un panel, es un cilindro puesto de pie y tumbado. A esto habrá que acostumbrarse al simplificar.

Además de los anteriores consejos positivos, hay que tener en cuenta que:

- nunca se puede formar un lazo con 3, 5, 6, etc. casillas, siempre con 2^n,
- nunca se pueden formar lazos en diagonal.

En la figura 3.6 se muestran algunos lazos en V-K de cuatro variables.

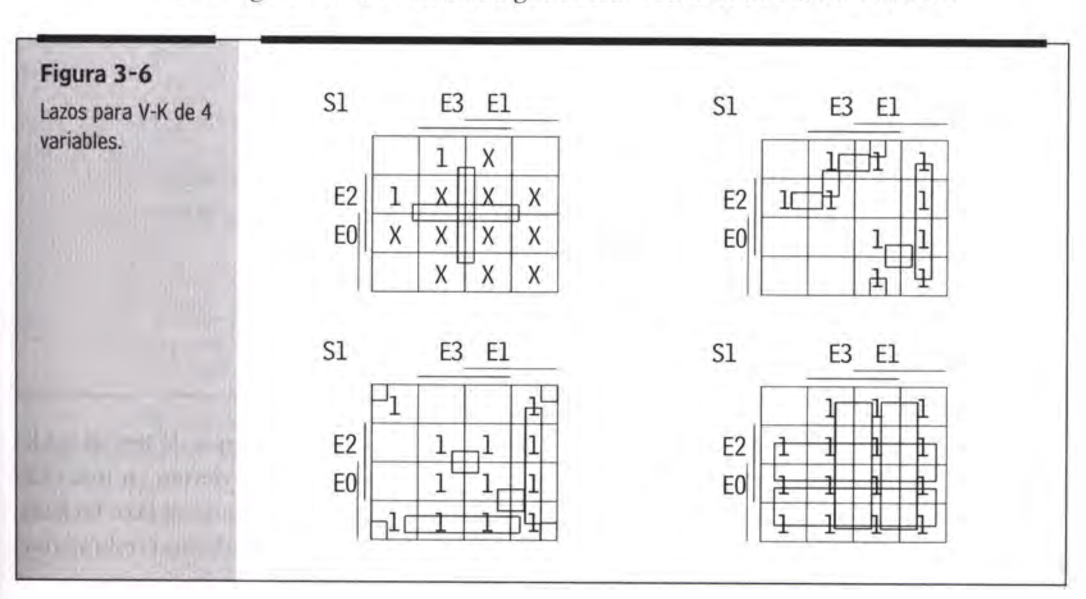


Figura 3-6
Lazos para V-K de 4 variables.

Simplificar exige ver los lazos del V-K, para elegir el menor conjunto de ellos que asegure cubrir la totalidad de 1 (0). Queda claro que es una habilidad que se desarrolla con la práctica.

Aunque generalmente cada persona simplifica de una manera, se puede enunciar un método que, bien seguido, asegura la minimidad de la expresión obtenida:

- formar todos los lazos de una casilla que no puedan formar lazos de dos,
- formar todos los lazos de dos casillas que no puedan formar lazos de cuatro,
- formar todos los lazos de cuatro casillas que no puedan formar lazos de ocho,
- etc.

Esta técnica, aunque es clara sobre el papel, a veces es confusa sobre el V-K. De hecho el método que muchos siguen, y que no asegura el acierto, es:

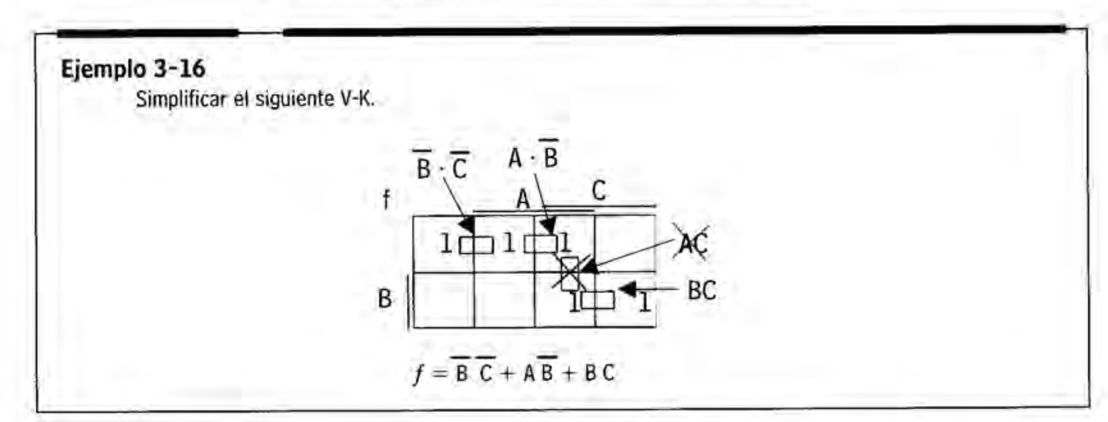
- formar lazos lo más grandes posible,
- rodear los 1 restantes con el menor número de lazos,
- repasar lo obtenido.

Ahora que sabemos escribir y formar los lazos de un V-K sólo nos falta ser capaces de obtener la expresión simplificada a partir de los lazos. Es decir, leer el V-K. Para leer cada lazo hay que fijarse en cómo están situadas las rayas de las variables respecto de los lazos:

- si la raya de una variable cubre todo el lazo, entonces la variable aparece positiva en el término,
- si la raya de una variable no cubre en absoluto el lazo, entonces la variable aparece negada en el término,
- si una raya cubre la mitad del lazo y la otra mitad no, entonces dicha variable no aparece en el término: ha sido simplificada.

Ejemplo 3-16

Simplificar el siguiente V-K.



$f = \overline{B}\,\overline{C} + A\overline{B} + BC$

En el ejemplo vemos que inicialmente hay cinco minitérminos de tres variables (10 AND y 3 OR de dos entradas) que al simplificar se convierten en una SOP de tres AND y dos OR. En el ejemplo se ve que no es necesario el lazo tachado, ya que los 1 por él rodeados ya lo están por otros lazos, resultando redundante; simplificar supone formar pocos lazos de gran tamaño.

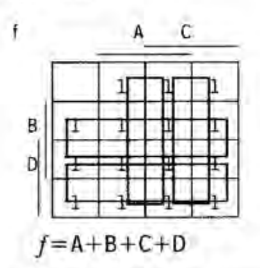


Simplificar es una habilidad que se obtiene con la práctica. A continuación presentaremos varios ejemplos que abundan en algunos consejos válidos a la hora de simplificar:

- no importa que los lazos se crucen entre sí (es lo que se busca),
- una casilla puede pertenecer a varios lazos (es lo que se busca),
- desterrar ciertos gustos estéticos a la hora de simplificar,
- resolver las situaciones incómodas con algo de ingenio y
- la función simplificada no tiene por qué ser única.

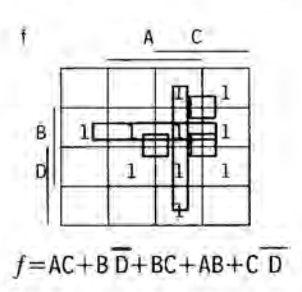
En los siguientes diagramas de V-K se muestran algunas situaciones singulares que ayudarán al lector a simplificar.

Formar lazos del mayor tamaño posible.



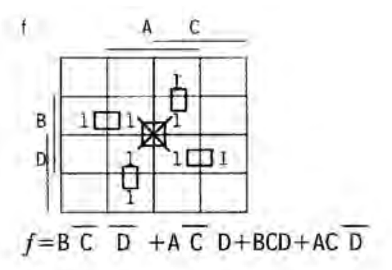
$f = A + B + C + D$

Una misma casilla puede formar parte de muchos lazos.



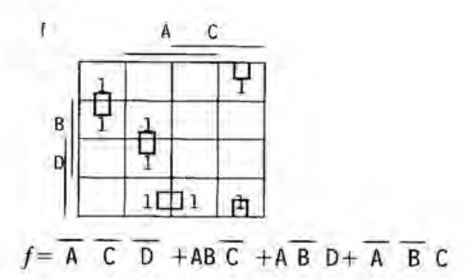
$f = AC + B\overline{D} + BC + AB + C\overline{D}$

Aplicar el criterio sin "gestos estéticos". Al formar los lazos de 2, el de 4 no es necesario (aunque puede que lo hayamos puesto el primero).

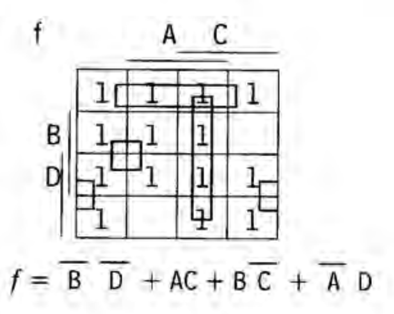


$f = B\overline{C}\,\overline{D} + A\overline{C}D + BCD + AC\overline{D}$

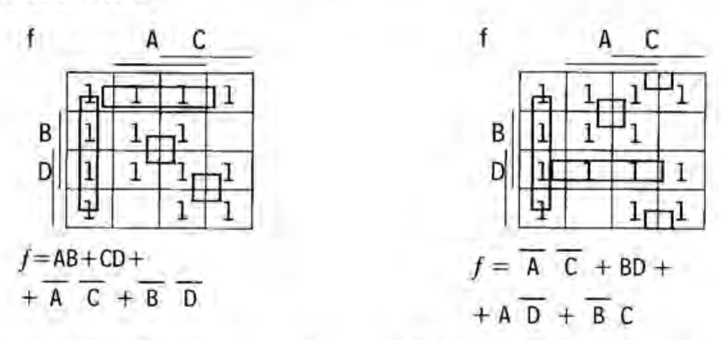
Aplicar el método con recursos. Si no hay ningún lazo por el que empezar, empezar por alguno. En este caso hay ocho 1, cualquier solución de 4 lazos de 2 es buena.



O para catorce 1 su mínimo rodeo son 4 lazos de 4.



La expresión no tiene por qué ser única. Las dos expresiones siguientes son equivalentes a la anterior.



Hasta el momento hemos visto cómo simplificar desde los 1 para obtener la expresión mínima SOP. Falta plantear cómo simplificar desde los 0 para obtener una expresión POS.

3.9.5 Simplificación con V-K desde los 0

El diagrama de V-K está pensado para simplificar desde los 1. Para simplificar desde los 0 se pueden hacer dos cosas:

- rehacer el V-K para favorecer la simplificación desde los 0, o
- modificar el método para hacer compatibles el V-K desde los 1 con la simplificación desde los 0.

Lo más útil es optar por la segunda estrategia.

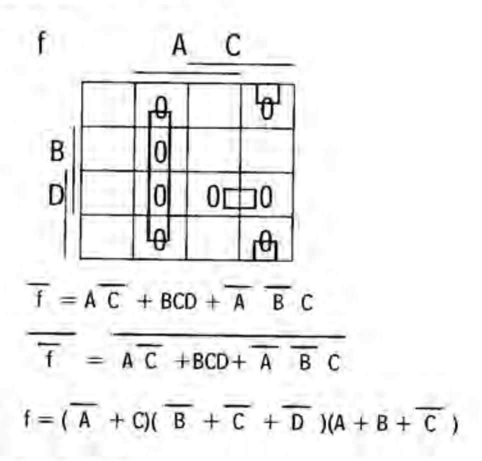
Para simplificar desde los ceros:

- Escribir el V-K desde los maxitérminos.
- Formar lazos con los 0 como si fueran 1.
- Escribir la suma de los anteriores lazos e igualarla a f negada.
- Aplicar el teorema de Morgan para obtener la expresión de f.

Es decir, trabajar como si fueran 1, pero como son ceros la expresión SOP es f negada. Al aplicar Morgan convertimos la SOP en POS. Así pues, simplificar desde los 0 es más largo, pero no más difícil.

Ejemplo 3-17

Obtener la expresión simplificada del siguiente V-K.



$\overline{f} = A\overline{C} + BCD + \overline{A}\,\overline{B}\,C$

$\overline{\overline{f}} = \overline{A\overline{C} + BCD + \overline{A}\,\overline{B}\,C}$

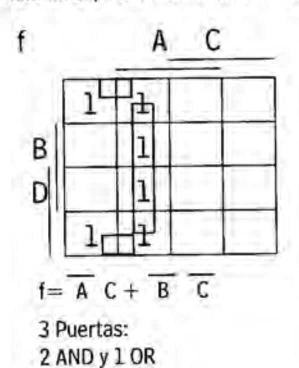
$f = (\overline{A} + C)(\overline{B} + \overline{C} + \overline{D})(A + B + \overline{C})$

Al simplificar desde los 0 aparecen tres cuestiones:

- ¿Son equivalentes las expresiones SOP y POS obtenidas? Por supuesto que sí, aunque a veces *la vista* nos haga creer lo contrario.
- ¿Son igual de mínimas la SOP que la POS? La respuesta es en general no.
- ¿Qué expresión es más económica, la SOP o la POS? A priori no se sabe, hay que obtener las dos y elegir la menor.

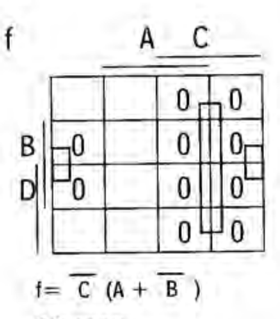
Ejemplo 3-18

Obtener la mínima expresión de la función representada por su V-K.



$f = \overline{A}\,C + \overline{B}\,\overline{C}$

3 Puertas:
2 AND y 1 OR



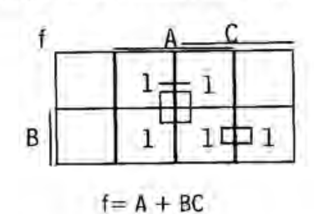
$f = \overline{C}\,(A + \overline{B})$

2 Puertas:
1 AND y 1 OR

En este caso vemos que la expresión POS obtenida desde los 0 es menor que la SOP obtenida desde los 1.

Ejemplo 3-19

Comprobar que las expresiones SOP y POS son equivalentes.



$f = A + BC$

$f = (A + B)(A + C)$

$f = (A + B)(A + C) = AA + AC + BA + BC = A + AC + AB + BC = A(1 + C + B) + BC =$
$= A \cdot 1 + BC = A + BC$

Por manipulación algebraica observamos que son equivalentes.

3.9.6. Simplificación con condiciones libres

Cuando en una tabla de verdad asignamos una X a una fila de la entrada, indicamos que, por la razón que sea, la salida puede ser 1 o 0.

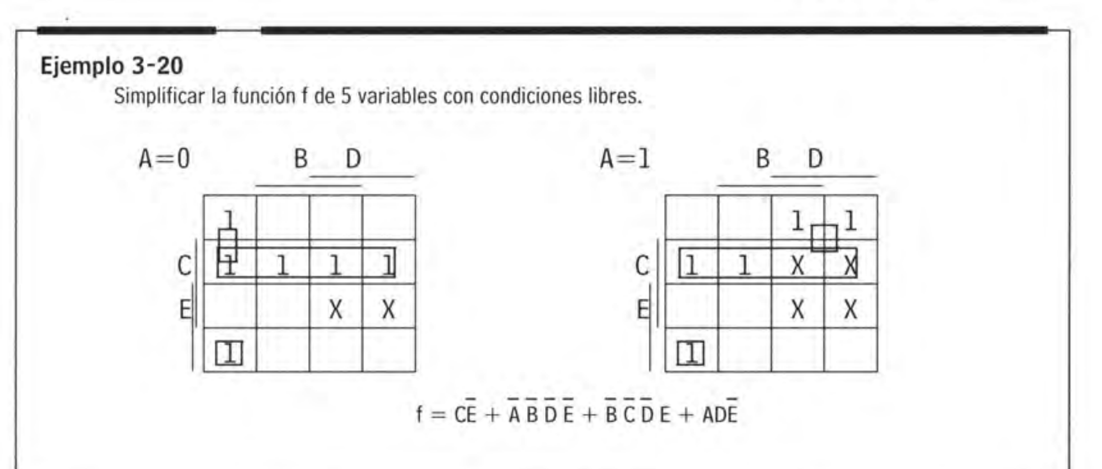
Desde el punto de vista de la simplificación una condición libre es un comodín. Las condiciones libres potencian la simplificación, pero también la complican.

Metodológicamente, a la hora de simplificar con X hay que tener en cuenta que:

- una X puede formar parte de un lazo o no, de forma libre,
- las X reducen el número de lazos o aumentan su tamaño,
- nunca se debe formar un lazo con sólo X y
- nunca se formará un lazo que se distinga de otro sólo en sus X.

Ejemplo 3-20

Simplificar la función f de 5 variables con condiciones libres.



$f = C\bar{E} + \bar{A}\,\bar{B}\,\bar{D}\,\bar{E} + \bar{B}\,\bar{C}\,\bar{D}\,E + AD\bar{E}$

3.10. Análisis y diseño con funciones booleanas

En este capítulo hemos visto una descripción básica del álgebra de Boole, y derivada de ella ciertos métodos y técnicas de representación. Este capítulo, si bien tiene una importancia teórica, es principalmente metodológico (por lo menos en un tipo de libro básico).

Visto así, podemos observar el capítulo como:

- una colección de formas de representación de funciones booleanas y
- una colección de métodos de transformación entre representaciones.

La figura 3.7 muestra la relación entre los distintos modos de representación. Cada flecha de la figura indica que se puede pasar de la representación origen a la de destino, aplicando el método correspondiente.

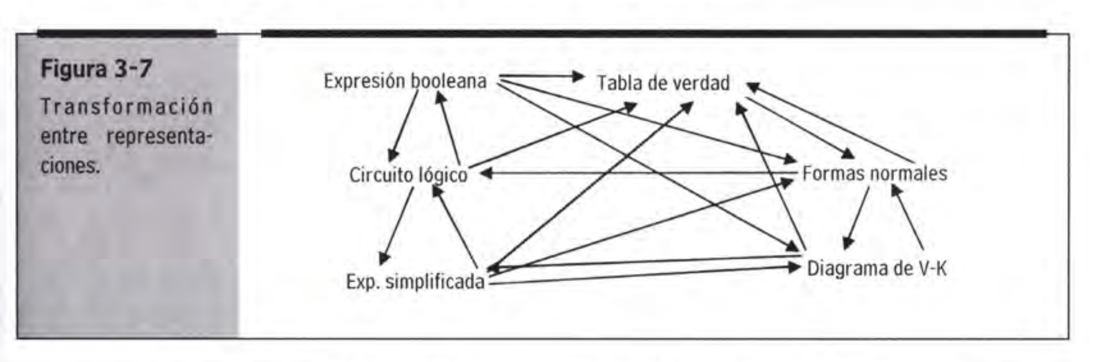


Figura 3-7 Transformación entre representaciones.

De hecho, diseñar y analizar sistemas digitales básicos es un proceso de transformación entre distintas representaciones de un mismo sistema. La figura 3.8 muestra qué es analizar (flechas discontinuas) y diseñar (flechas continuas).

La figura no viene sino a reforzar la importancia de los distintos métodos, tan sencillos como potentes y radicales en su aplicación. Este capítulo ha mostrado con claridad cómo aplicarlos y cómo encajarlos en el edificio teórico de la electrónica digital.

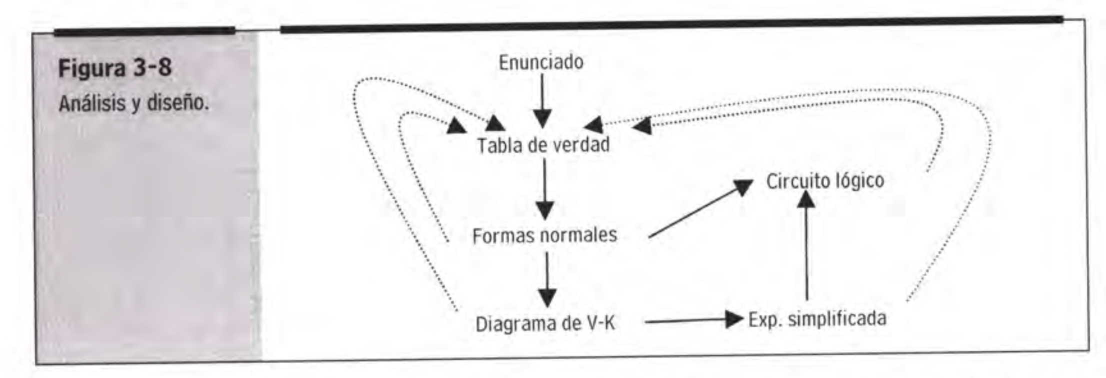


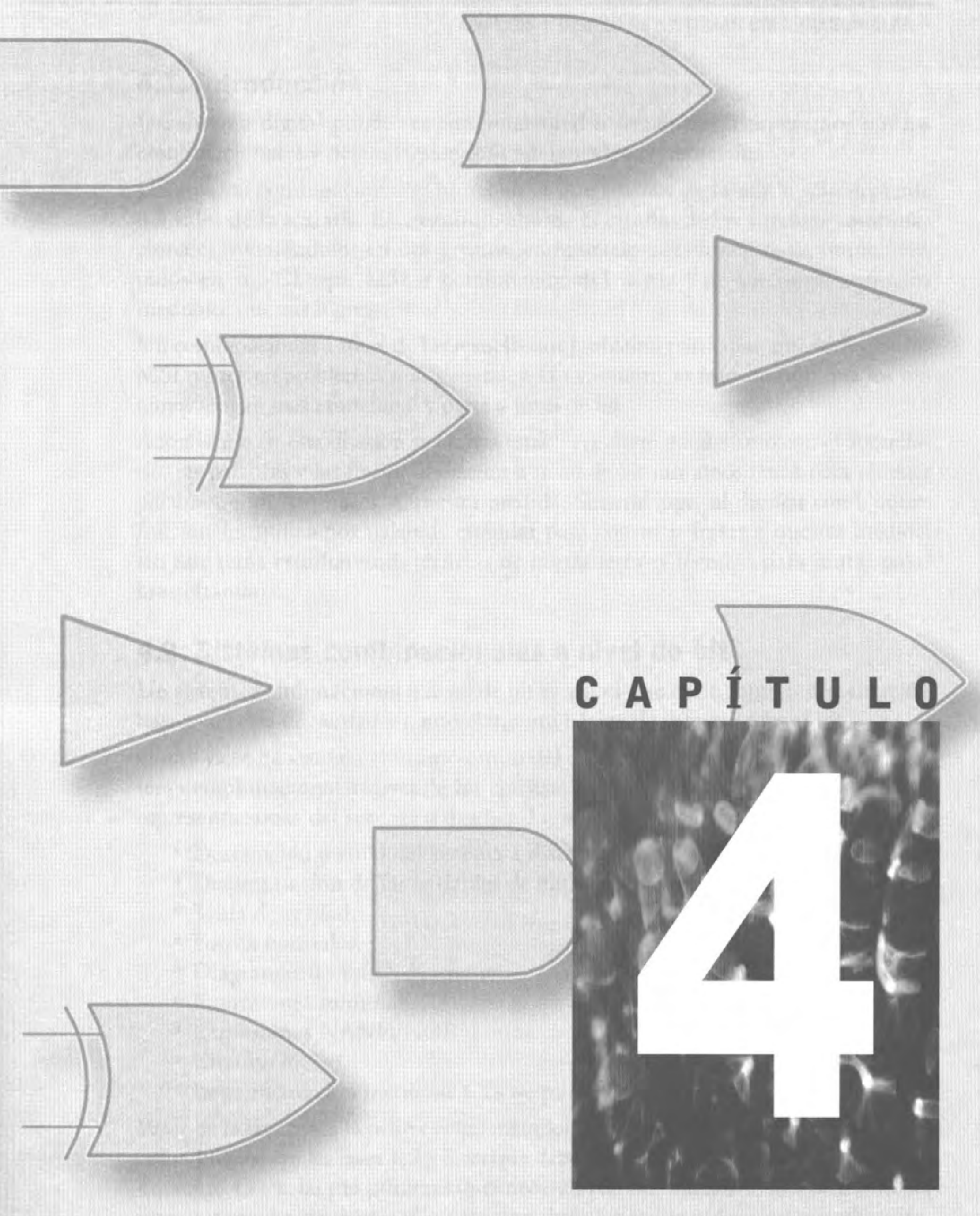
Figura 3-8
Análisis y diseño.

Al acabar este capítulo el lector debe tener la sensación de que dispone de una caja de herramientas básicas para comenzar a analizar y diseñar sistemas digitales.

3.11. Resumen

Este capítulo es de una importancia capital para el desarrollo del libro. En él se han introducido las bases teóricas y metodológicas del análisis y diseño a plantear en los restantes capítulos del libro.

En todo momento se ha utilizado un nivel que permita acercarse al álgebra de Boole a cualquier persona con algún conocimiento de expresión matemática, evitando discusiones teóricas. El lector que quiera profundizar en el álgebra de Boole encontrará mucho material ya publicado.



CAPÍTULO 4

ANÁLISIS Y DISEÑO DE SISTEMAS COMBINACIONALES

4.1. Introducción

Un sistema digital puede ser combinacional o secuencial. Empecemos por los combinacionales y dejemos para más adelante los secuenciales.

Un sistema combinacional es aquel en el que el valor de la salida sólo depende del valor de la entrada. Este capítulo aborda el estudio de los sistemas combinacionales dividiéndolos en dos grupos: combinacionales funcionales implementados en un CI tipo MSI y combinacionales a nivel de bit implementados mediante puertas lógicas.

Un combinacional a nivel de bit resuelve un problema particular, mientras que los MSI resuelven problemas más generales. Por supuesto, es muy común que los sistemas tengan parte funcional y parte a nivel de bit.

Ahondando en esta división podemos establecer cierto paralelismo con el abecedario, las palabras y las frases: al diseñar a nivel de bit juntamos letras para obtener palabras, e incluso frases, a nuestra medida. Sin embargo, al diseñar con bloques funcionales utilizamos palabras estándar para construir frases a nuestra medida. En este tema estudiaremos técnicas de juntar letras y técnicas para juntar palabras estándar.

4.2. Sistemas combinacionales a nivel de bit

Un sistema combinacional a nivel de bit es aquel que tiene como representación básica la tabla de verdad y como elemento básico de información el bit.

Como ya se ha visto en el último punto del capítulo anterior, el diseño de un sistema combinacional a nivel de bit consiste en transformar entre sí las distintas representaciones del sistema a diseñar. La secuencia de transformaciones es:

- Descripción textual del sistema a diseñar.
- Determinación de las variables de entrada y salida.
- Tabla de verdad.
- Forma normales.
- Diagramas de Veitch-Karnaugh.
- Expresiones mínimas.
- Expresiones NAND/NOR u otras.
- Circuito lógico.
- Implementación mediante CI's de puertas lógicas.

Pasar de la fase tres a la ocho es algo metodológico que será explicado en este apartado; difíciles son las fases 1, 2 y 3, ya que dependen de nuestra capacidad de discernimiento. O sea, las tres primeras fases necesitan de cierta creatividad u originalidad, mientras que las siguientes sólo necesitan de una correcta aplicación metodológica.

La calidad de una descripción textual puede variar mucho según sea el emisor. Sus dos principales defectos son que puede ser incompleta y/o ambigua, es decir, quedan situaciones sin describir y/o algunas no lo están de forma clara. Sin embargo, la tabla de verdad es completa y no ambigua, y por tanto al completarla deberían solucionarse de manera adecuada las posibles deficiencias de la descripción textual.

Para cumplir el resto de los pasos basta con aplicar con rigor los métodos vistos en el capítulo anterior.

4.2.1. Uso de la tabla de verdad

A la hora de completar la tabla de verdad es útil pensar que cada fila de ésta es una situación o pregunta que se le plantea al sistema, a la cual debemos contestar de forma clara para cada salida: 0 o 1. Además de 1 y 0, existe una tercera posibilidad: la condición libre, que debe ser tratada con especial cuidado.

Al hablar de condiciones libres hay que abordar cuatro aspectos:

- Uso de las condiciones libres.
- Condiciones libres en la salida.
- Condiciones libres en la entrada.
- Efecto de las condiciones libres.

Cuando en la columna salida de una tabla de verdad aparece una condición libre 'X', ésta aparecerá en el correspondiente diagrama V-K, lo que facilitará y potenciará el proceso simplificador de dicha salida. Es decir, la condición libre se comporta como un comodín, y en principio es beneficiosa, pues agiliza la simplificación. Sin embargo es necesario matizar en qué condiciones es beneficiosa -condiciones libres en la salida-, y en cuáles puede ser peligrosa -condiciones libres en la entrada-.

En algunos casos, utilizar una condición libre en el diseño supone asumir unos riesgos, pues si se diera en la entrada la combinación considerada imposible o libre, resultaría que la salida tomaría un valor que a priori resultaría desconocido para nosostros. Es decir, esperamos que cierta situación no se dé, pero si se diera resultaría que hemos perdido el control de la salida. No es necesario ahondar en el tema para darse cuenta de las ventajas y peligros de las condiciones libres: su uso dependerá del estilo del diseñador y sobre todo del propio sistema a diseñar.

4.2.2. Un ejemplo

A continuación presentamos un ejercicio muy sencillo que nos permite plantear un gran número de cuestiones relativas al diseño de cualquier sistema digital.

Enunciado

Diseñar el circuito que mediante un indicador señale cuándo se debe ir al cine (C=1) y cuándo no (C=0) según la siguiente regla: si llueve y hace frío o nieva ir al cine.

Determinación de entradas y salidas

El primer paso consiste en determinar qué señales son de entrada y cuáles de salida. Para este ejemplo son entradas las variables lluvia (LL), frío (F) y nieve (N) y salida es cine (C). De una correcta determinación de las entradas y salidas dependerá el curso del diseño.

Tabla de Verdad del sistema

A la hora de completar la tabla de verdad plantearemos tres salidas distintas, fruto de las dudas que plantea el ejercicio (por simple que parezca).

Recordemos que cada fila de la tabla de verdad es una situación que se le presenta al sistema. Así, la primera fila 0 0 0 refleja que no llueve, no hace frío y no nieva y por tanto, y según nuestro conocimiento del sistema, no se debe activar la salida C. Este proceso se repite sucesivamente para todas las filas, pero algunas de ellas pueden ser tratadas de distinta forma, lo que nos lleva a plantear tres posibles y excluyentes salidas.

LL	F	N	C1	C2	C3
0	0	0	0	0	0
0	0	1	1	X	0
0	1	0	0	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	1	X	0
1	1	0	1	1	1
1	1	1	1	X	0

Si miramos a la fila 1 0 1 podemos decir que C1=1, ya que nieva, pero si la observamos con más cuidado vemos que 1 0 1 significa que nieva, ¡y además llueve sin hacer frío! Ante esta situación ilógica podemos optar por C=0, 1 o X, según lo entendamos o se derive del enunciado, resultando distintas columnas de salida, y por tanto distintas posibles ecuaciones a implementar.

En este caso la variable de salida es CINE; pero para imaginar la importancia de un planteamiento riguroso en el uso y determinación de las condiciones libres baste pensar en el accionamiento de un motor, una válvula, una alarma, etc.

Diseño lógico

Una vez planteada la tabla de verdad (utilizaremos C1), sólo resta plantear el correspondiente diagrama de V-K, simplificarlo e implementarlo, como se muestra en el diagrama 4-1 y en la figura 4-1, respectivamente.

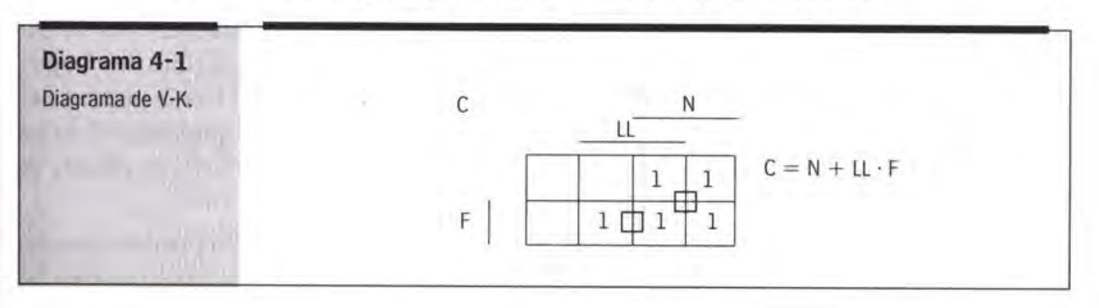


Diagrama 4-1
Diagrama de V-K.

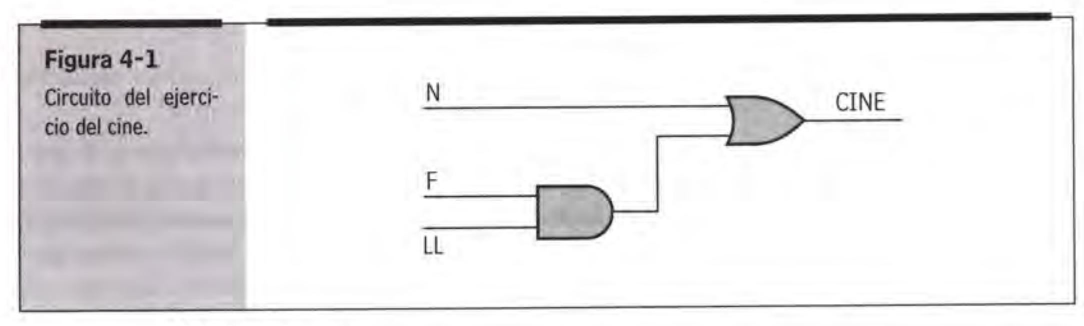


Figura 4-1
Circuito del ejercicio del cine.

De otra manera

Además, en este ejemplo no hubiera sido necesario el total seguimiento de los pasos, hubiera bastado con reescribir buenamente el enunciado: si llueve y hace frío o nieva voy al cine.

si llueve	y	hace frío	o	nieva		cine
↓	↓	↓	↓	↓		
(LL	·	F)	+	N	=	CINE

En el apartado 4.13 se resuelven distintos diseños combinacionales.

4.3. Circuitos combinacionales a nivel de palabra

Dentro del conjunto de los infinitos sistemas combinacionales planteables algunos de ellos resultan muy usuales, tanto que son implementados de forma particular en un único CI con tecnología MSI (Medium Scale Integration), convirtiéndose en estándares. A estos bloques se les denomina de muchas maneras, pero quizá la más adecuada sea la de *bloques funcionales*.

Aunque en principio es un grupo heterogéneo, los bloques funcionales MSI tienen ciertas características comunes:

- Una funcionalidad muy clara.
- Uso de palabras binarias y no bits.
- Extensibilidad.
- Funcionalidad auxiliar.

En un combinacional MSI la información no se procesa a nivel de bit, sino a nivel de grupo de bits, de palabra. Así, la entrada será un dígito BCD, o un código ASCII, o un grupo de cuatro u ocho bits. La función que implementa el MSI contempla la entrada en su conjunto, no bit a bit.

La extensibilidad permite obtener un circuito de complejidad MxN con M circuitos de complejidad N, sin necesidad de diseñar *ad hoc* el circuito de complejidad MxN. O sea, con M circuitos de longitud de palabra N bits se puede obtener un circuito de longitud MxN bits. Esta característica es fundamental en los MSI, y entronca directamente con el tratamiento de datos a nivel de palabra de tamaño extensible según sean las necesidades del sistema.

Además, hay que destacar que algunos circuitos MSI no sólo pueden comportarse según su definición, sino también cubrir otras funciones auxiliares para las que no fueron diseñados.

Los circuitos combinacionales MSI descritos en este capítulo son:

- Codificadores/Decodificadores.
- Multiplexores/Demultiplexores.
- Generadores/Detectores de paridad.
- Comparadores.
- Conversores de códigos.

En los combinacionales funcionales las señales pueden ser de dos tipos:

- De Entrada/salida. Condicionan el comportamiento del circuito.
- De Control. Controlan el desarrollo de las operaciones y su resultado:
  - ENABLE INPUT.
  - ENABLE OUTPUT.
  - GROUP SIGNAL.
  - INHIBITION.
  - Etc.

Al ser descritos, aunque cada uno de estos circuitos tiene sus particularidades, para todos ellos seguiremos un mismo guión que facilite su estudio:

- definición textual,
- entradas y salidas,
- tabla de verdad funcional,
- ecuaciones booleanas,
- circuito lógico,
- bloque funcional,
- extensibilidad,
- circuitos MSI
- ejemplos y
- casos particulares.

Respecto del diseño, ya podemos decir desde ahora que no existe una metodología estricta de diseño, sino técnicas y experiencia.

A la hora de seguir leyendo el capítulo (e incluso el resto del libro) el lector puede tomar dos actitudes:

- Visión externa: el lector sólo se interesa por el qué hace el circuito y para qué lo hace. El lector toma una visión utilitaria.
- Visión interna: además del anterior punto de vista, el lector también se interesa por el cómo lo hace el circuito. El lector toma un punto de vista más didáctico y completo.

Un camino no es mejor que otro (y pueden alternarse), dependerá de las necesidades de cada lector.

4.4. Codificadores

Un codificador es un dispositivo que transforma una señal expresada en un *código humano* -código 1 entre M- en una señal expresada en un código binario -en binario puro mientras no se indique otro código-. Por ejemplo, codificar las señales de un teclado decimal en BCD (figura 4-2).

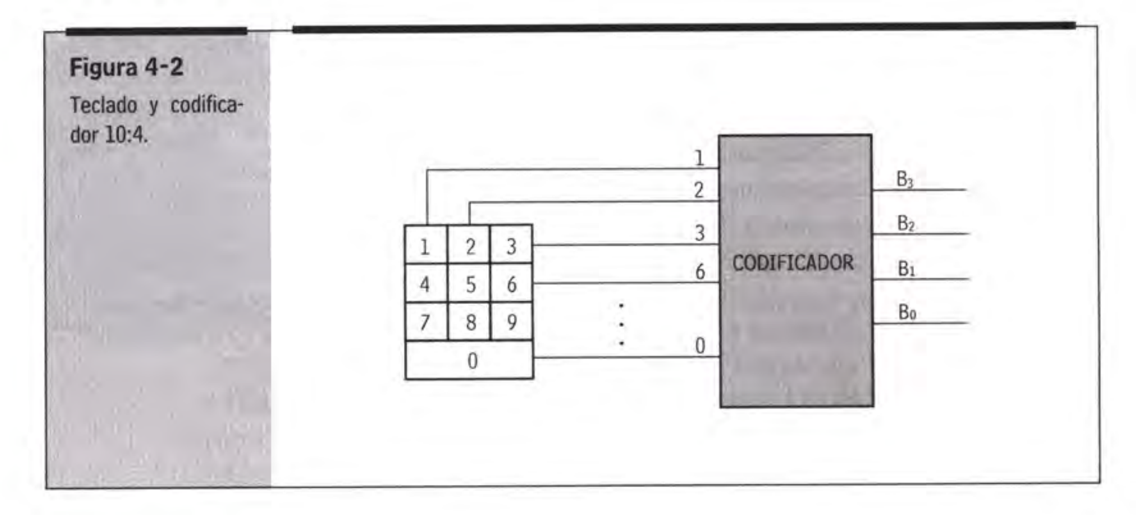


Figura 4-2
Teclado y codificador 10:4.

Definición

Es un circuito con m entradas y n salidas, representado como *m:n* o también como *m a n*. A cada línea de entrada se le asigna un peso. Si se activa una línea de entrada, en los n bits de salida aparece codificado -en el código elegido- el peso de la entrada.

Un codificador se denomina "*completo*" si $m=2^n$; en caso contrario, si $m<2^n$, el codificador es incompleto. Por ejemplo los codificadores 8:3 y 4:2 son completos, mientras el 10:4 es incompleto.

Los codificadores se dividen en dos tipos según atiendan a la simultaneidad en la activación de las líneas de entrada:

- Codificadores sin prioridad.
- Codificadores con prioridad.

4.4.1. Codificadores sin prioridad

Este tipo de codificador está diseñado para atender sólo una activación en las líneas de entrada simultáneamente, es decir, sólo se debe activar una línea de entrada en cada instante. O dicho de otra forma, si se activara más de una línea de entrada la salida obtenida no tendría por qué ser correcta -que no es lo mismo que ser errónea, pero casi-.

4.4.1.1. Codificador 4:2 sin prioridad

Diseñemos un codificador 4:2 en binario puro sin prioridad (mientras no se diga lo contrario el código siempre será el binario puro). En la tabla de verdad -tabla 4-1- la activación simúltanea de más de una entrada no aparece en la tabla y es considerada imposible, y por tanto se le asocia una condición libre.

Tabla 4-1
T-V codificador 4:2 sin prioridad.

A3	A2	A1	A0	S1	S0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

De la tabla de verdad anterior se obtienen los diagramas de V-K del diagrama 4-2.

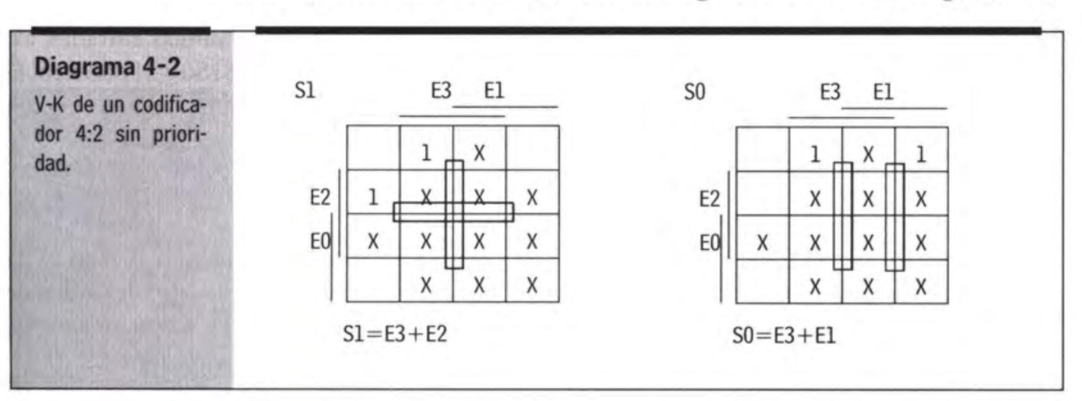


Diagrama 4-2
V-K de un codificador 4:2 sin prioridad.

Las ecuaciones booleanas simplificadas que describen el comportamiento de un codificador 4:2 sin prioridad son las de 4.1. La implementación aparece en la figura 4-3.

$$S_1 = E_3+E_2$$
$$S_0 = E_3+E_1 \qquad (4.1)$$

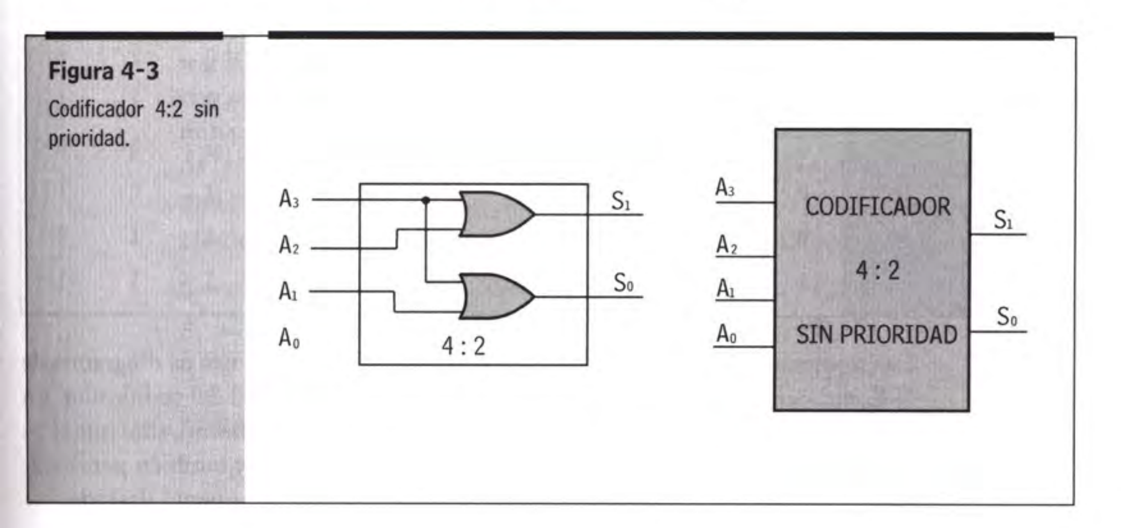


Figura 4-3
Codificador 4:2 sin prioridad.

A la vista del diseño anterior podemos destacar que la línea de entrada A0 no participa en el circuito, eso no quiere decir que sea despreciable, todo lo contrario: la salida asociada al codificador en reposo es la correspondiente a la entrada A0. Es decir, este circuito por defecto se comporta como si tuviera activada la entrada A0.

También es observable la sencillez y claridad de las expresiones obtenidas, por ejemplo: la línea de salida S1 se activará cuando lo estén la entrada A3 o la entrada A2. Este comportamiento es fácilmente extensible a codificadores de mayor capacidad, sin más que tener en cuenta el código binario puro.

Veamos un caso de mal funcionamiento: si por ejemplo se activara A1 en la salida tendríamos S1S0=01, si fuera A2 tendríamos S1S0=10, pero si se activaran, aunque no se deba, A1 y A2, la salida reflejaría el efecto de ambas entradas, así S1=A3+A2=0+1=1 y S0=A3+A1=0+1=1 y por tanto S1S0=11, es decir, la salida se comportaría como si se hubiera activado la entrada A3, dando lugar a una situación incorrecta, o mejor dicho, inconsistente con la realidad.

4.4.1.2. Codificador 8:3 sin prioridad

Al plantearnos el diseño de un codificador 8:3 en binario puro y un 10:4 en BCD podríamos repetir el proceso anterior, pero tendríamos que manejar del orden de 8 y 10 entradas, o sea, 256 o 1.024 filas en la tabla de verdad. Pero esto no es necesario si extendemos el comportamiento observado para el codificador 4:2.

La tabla de verdad correspondiente a un codificador 8:3 sin prioridad es la tabla 4-2.

Tabla 4-2
T-V codificador 8:3 sin prioridad.

A7	A6	A5	A4	A3	A2	A1	A0	S2	S1	S0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Las siguientes ecuaciones no han sido obtenidas mediante el uso de diagramas de V-K, etc., sino de forma intuitiva, afirmando la extensibilidad del codificador. En este caso la tabla de verdad no cumple un papel activo en el diseño, sino que sirve para representar y describir el comportamiento del sistema, y también para comprobar que las ecuaciones planteadas responden al funcionamiento deseado.

$$S_2 = E_4+E_5+E_6+E_7$$
$$S_1 = E_2+E_3+E_6+E_7 \qquad (4.2)$$
$$S_0 = E_1+E_3+E_5+E_7$$

Si leyéramos con detenimiento las ecuaciones 4.2 veríamos que dicen que la salida de peso 4 (S2) se pone a 1 sólo si está activa la entrada E4, la E5, la E6 o la E7; lo que es correcto. Lo mismo para S0 (peso 1) y las entradas E1, E3, E5 y E7; justo las impares.

Generalizando, podemos decir que las ecuaciones de cada salida de un codificador sin prioridad se obtienen sumando las entradas activas para cada 1 de cada salida; lo que podemos comprobar con la tabla 4-2.

La figura 4-4 es el circuito lógico del codificador 8:3.

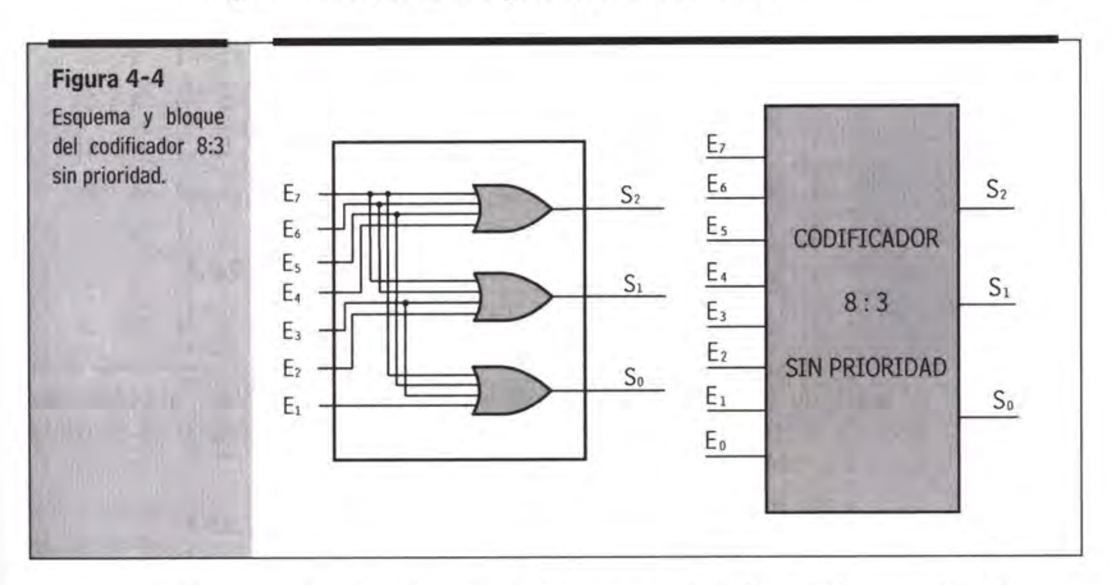


Figura 4-4
Esquema y bloque del codificador 8:3 sin prioridad.

Veamos sobre el codificador 8:3 el uso y significado de la línea auxiliar de control EI (Enable Input). Si EI estuviera activada la salida reflejaría el estado de la entrada; pero si no estuviera activada, la salida quedaría en un estado predeterminado, es decir, aunque se activara alguna entrada su efecto no pasaría a la salida (es como si EI fuera una *llave*, cuya activación previa es necesaria para el normal funcionamiento).

Desde un punto de vista booleano es una simple AND:

$$S_2 = E_7 \cdot EI + E_6 \cdot EI + E_5 EI + E_4 \cdot EI = (E_7+E_6+E_5+E_4)EI$$
$$S_1 = E_7 \cdot EI + E_6 \cdot EI + E_3 EI + E_2 \cdot EI = (E_7+E_6+E_3+E_2)EI$$
$$S_0 = E_7 \cdot EI + E_5 \cdot EI + E_3 EI + E_1 \cdot EI = (E_7+E_5+E_3+E_1)EI$$

Si EI=0 todas las salidas quedarán a cero, si EI=1 las salidas *seguirán* en su comportamiento a las entradas.

4.4.1.3. Codificador 10:4 sin prioridad

Diseñemos el codificador 10:4 teniendo en cuenta la línea EI e introduciendo las también líneas de control E0 y GS.

La tabla de verdad correspondiente a un codificador 10:4 sin prioridad es la tabla 4-3.

Tabla 4-3
Tabla de verdad de un codificador 10:4 sin prioridad.

EI	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	S3	S2	S1	S0	EO	GS
0	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1
1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1
1	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1
1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	1
1	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	1
1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	1
1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1

Al igual que para 4.2, las ecuaciones booleanas que representan al codificador 10:4 son obtenidas intuitivamente en 4.3, pudiéndose comprobar su exactitud frente a la tabla de verdad.

$$
\begin{aligned}
S_3 &= (E_9+E_8)EI \\
S_2 &= (E_7+E_6+E_3+E_2)EI \\
S_1 &= (E_9+E_7+E_5+E_3+E_1)EI \\
EO &= EI \cdot \overline{E_9} \cdot \overline{E_8} \cdot \overline{E_7} \cdot \overline{E_6} \cdot \overline{E_5} \cdot \overline{E_4} \cdot \overline{E_3} \cdot \overline{E_2} \cdot \overline{E_1} \cdot \overline{E_0} \\
GS &= EI(E_9+E_8+E_7+E_6+E_5+E_4+E_3+E_2+E_1+E_0)
\end{aligned}
\qquad (4.3)
$$

La figura 4-5 es el circuito lógico del codificador 10:4.

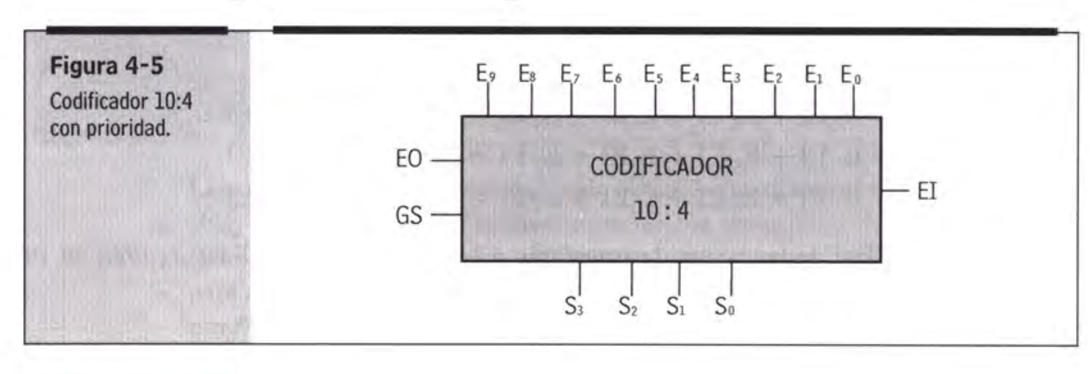


Figura 4-5
Codificador 10:4 con prioridad.

Existen 3 situaciones en las que $S_3S_2S_1S_0$=0000:

- cuando se activa la entrada A0,
- cuando no se activa entrada alguna y
- cuando está desactivada EI.

Es decir, distintas situaciones en la entrada generan una misma salida. Este comportamiento claramente no es deseable, aunque en muchos casos no es preocupante. Las líneas E0 (Enable Output) y GS (Group Selection) nos ayudan a diferenciarlas:

- E0 se activa cuando estando EI activa no se active entrada alguna.
- GS se activa cuando estando EI activa se active alguna entrada.

Teniendo en cuenta EI, GS y EO podemos distinguir los tres casos anteriores donde $S_{3\text{-}0}$ = 0000. Además, como se verá más adelante, E0, GS y EI permiten la conexión de distintos codificadores entre sí para conformar codificadores de mayor capacidad.

Volviendo a los casos extraños, si en este codificador 10:4 se activaran simultáneamente E9 y E6, según las ecuaciones aportadas la salida sería $S_3S_2S_1S_0$=1111, resultado totalmente erróneo. Para evitar estas situaciones se utiliza un codificador con prioridad.

4.4.2. Codificador con prioridad

Este codificador contempla y da respuesta adecuada a la posible activación de varias entradas simultáneamente. En esta situación es necesario determinar el criterio que determine qué entrada tiene prioridad. El criterio más común es: se dará prioridad a la línea de más peso, aunque bien pudiera ser cualquier otro criterio.

4.4.2.1. Codificador 4:2 con prioridad

Diseñemos un codificador 4:2 con prioridad. La tabla de verdad correspondiente es la tabla 4.4, y en este caso se contemplan y resuelven las situaciones en las que se activa más de una entrada.

Tabla 4-4
T-V codificador 4:2 con prioridad.

EI	E3	E2	E1	E0	S1	S0	EO	GS
0	X	X	X	X	0	0	0	0
1	0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	0	1
1	0	0	1	X	0	1	0	1
1	0	1	X	X	1	0	0	1
1	1	X	X	X	1	1	0	1

Diagrama 4-3
V-K de un codificador 4:2 con prioridad.

S1 (E3, E1)

		1	1	
E2	1	1	1	1
E0	1	1	1	1
		1	1	1

S1=E3+E2

S0 (E3, E1)

		1	1	1
E2		1	1	
E0		1	1	
		1	1	1

$S0=E3+\overline{E2}E1$

Las ecuaciones 4.4 han sido simplificadas en los diagramas de V-K 4-3. Al añadir la línea EI, resultan las ecuaciones 4.4, y de ellas el bloque de la figura 4-6.

$$S_1 = EI(E_3+E_2) \qquad S_0 = EI(E_3+\overline{E_2}\ E_1)$$
$$EO = EI\ \overline{E_3}\ \overline{E_2}\ \overline{E_1}\ \overline{E_0}$$
$$GS = EI(E_3+E_2+E_1+E_0) \qquad (4.4)$$

En este caso si se activaran E_1 y E_2, sólo el efecto de E_2 pasaría a la salida pues:

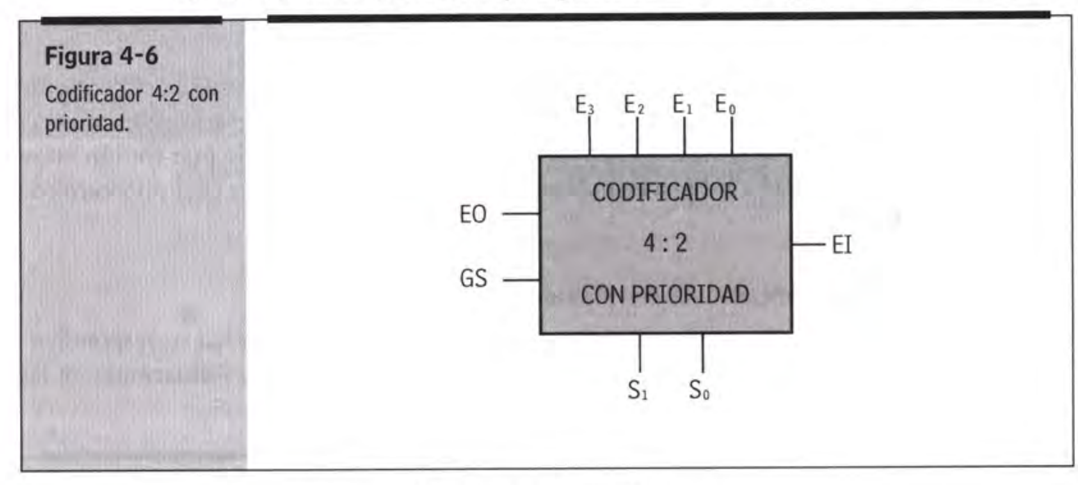


Figura 4-6
Codificador 4:2 con prioridad.

$$S_1 = EI(E_3+E_2) = 1\cdot(0+1) = 1,$$
$$S_0 = EI(E_3+\overline{E_2}\ E_1) = 1\cdot(0+0\cdot1) = 0$$
$$S_1\ S_0 = 10$$

Justo lo correspondiente a E_2.

En cuanto a las expresiones, éstas son algo más complicadas pero igualmente claras: para que E_1 muestre su efecto en la salida es necesario que E_3 y E_2 estén inactivas, $\overline{E_3}$.y $\overline{E_2}$, y así sucesivamente para todas las entradas.

4.4.2.2. Codificador 8:3 con prioridad

Pasemos a describir el codificador 8:3 comercial 74148 con prioridad (tabla 4-5, ecuaciones 4.5 y figura 4.7) en el que tanto las entradas como las salidas y las líneas EI, E0 y GS son activas por nivel bajo. Esto quiere decir que para $\overline{E6}$=0 y $\overline{EI}$=0 las salidas son $\overline{S3}$ $\overline{S2}$ $\overline{S1}$ $\overline{S0}$= 1001, $\overline{EO}$= 1 y $\overline{GS}$= 0. En este caso, y sólo por simbolismo, las líneas pasan a llamarse negadas, por ejemplo EI pasa a $\overline{EI}$.

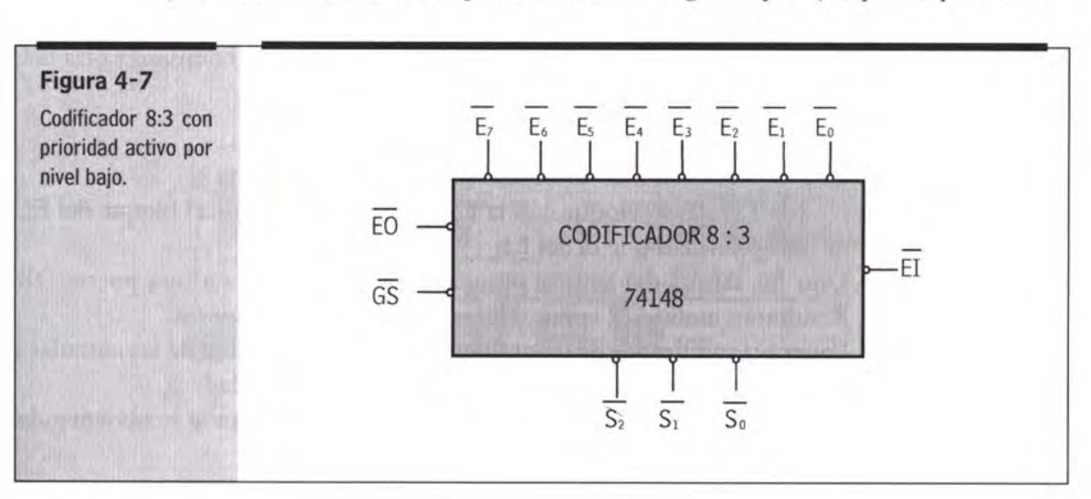


Figura 4-7
Codificador 8:3 con prioridad activo por nivel bajo.

Tabla 4-5
Tabla de verdad del codificador 8:3 74148.

$\overline{EI}$	$\overline{E7}$	$\overline{E6}$	$\overline{E5}$	$\overline{E4}$	$\overline{E3}$	$\overline{E2}$	$\overline{E1}$	$\overline{E0}$	$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	$\overline{EO}$	$\overline{GS}$
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	0	1	1	1	1	0
0	1	1	1	1	1	1	0	1	1	1	0	1	0
0	1	1	1	1	1	0	1	1	1	0	1	1	0
0	1	1	1	1	0	1	1	1	1	0	0	1	0
0	1	1	1	0	1	1	1	1	0	1	1	1	0
0	1	1	0	1	1	1	1	1	0	1	0	1	0
0	1	0	1	1	1	1	1	1	0	0	1	1	0
0	0	1	1	1	1	1	1	1	0	0	0	1	0

$$\begin{aligned}
\overline{S2} &= \overline{\overline{EI}\,(\overline{E7} + \overline{E6} + E6\,\overline{E5} + E6E5\,\overline{E4})} \\
\overline{S1} &= \overline{\overline{EI}\,(\overline{E7} + \overline{E6} + E6E5E4\,\overline{E3} + E6E5E4E3\,\overline{E2})} \qquad (4.5) \\
\overline{S0} &= \overline{\overline{EI}\,(\,\overline{E7} + E6\,\overline{E5} + E6E5E4\,\overline{E3} + E6E5E4E3E2\,\overline{E1})} \\
\overline{GS} &= \overline{\overline{EI}\,(\,E7 + E6 + E5 + E4 + E3 + E2 + E1 + E0)}
\end{aligned}$$

4.4.3. Extensión de la capacidad de un codificador

En general, en el mercado sólo hay codificadores con un máximo de 10 entradas. Si el sistema necesitara de más entradas la solución pasaría por interconectar codificadores entre sí, utilizando las líneas EI, EO y GS (activas por nivel alto o bajo).

Existen diversos métodos de extender la capacidad de un codificador, alguno de ellos tan enrevesado como útil. A continuación veremos dos técnicas: una más teórica y general que servirá para enfocar el problema en su conjunto y otra más práctica.

En un método ordenado los pasos a dar son:

- Elegir el número n de codificadores necesarios.
- Unir la EO de un bloque con la EI del siguiente, siendo el bloque del EO de más prioridad que el del EI.
- Unir las salidas del mismo peso de cada codificador en una puerta OR. Resultarán tantas OR como salidas tengan los codificadores.
- Poner un codificador de n entradas, conectando cada una de las entradas a las líneas GS de cada codificador respetando la prioridad.
- Obtener la salida teniendo en cuenta que las de más peso se obtienen del codificador y el resto de las puertas OR.

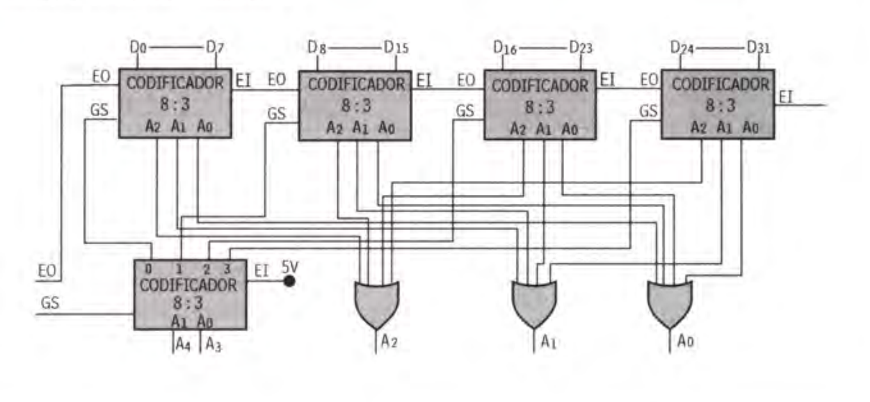
Ejemplo 4-1

Diseñar un codificador 16:4 con codificadores 4:2.

Respetando los anteriores pasos nos hacen falta cuatro codificadores 4:2, dos puertas OR de cuatro entradas y un último codificador 4:2. El esquema resultante es el de la figura 4-8.

Figura 4-8

Codificador 32:5 con codificadores 8:3.



 © ITES - PARANINFO

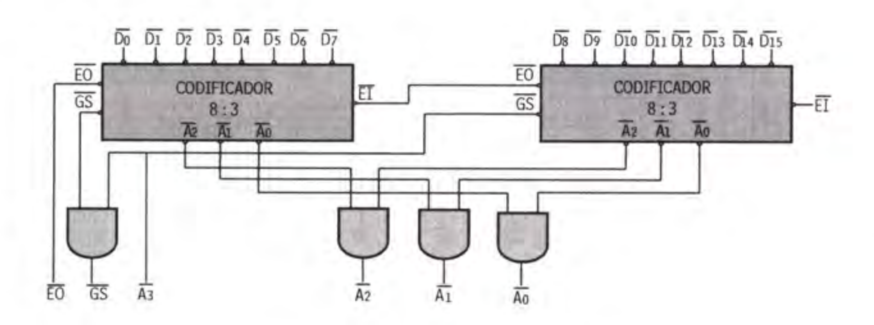
Ejemplo 4-2

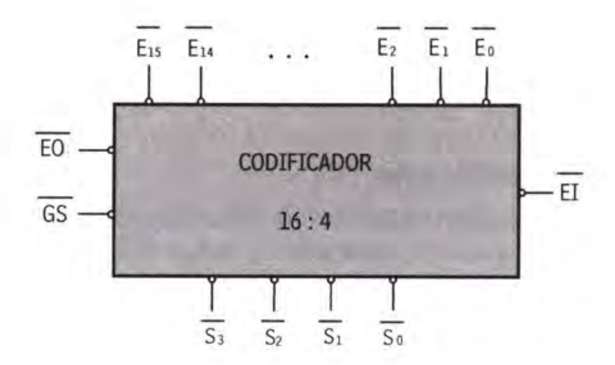
Diseña un codificador 16:4 con codificadores 8:3.

La figura 4-9 plantea el circuito que se comporta como un codificador 16:4 con las líneas activas por nivel bajo. Véase cómo en este caso el nivel bajo en entradas y salidas convierte las OR en AND, y cómo debido al tamaño 16:4 no hace falta el codificador en la salida; es la propia señal GS.

Figura 4-9

Conexión de dos 8:3 para obtener un codificador 16:4





Si observamos el esquema 4-9 vemos que la línea $\overline{EO}$ es conectada a $\overline{EI}$ según el razonamiento: si ninguna entrada entre $\overline{D8}$ y $\overline{D15}$ ha sido activada (las de mayor prioridad) $\overline{EO}$ se pone a cero, y por tanto, al estar conectada a $\overline{EI}$, se habilita el servicio a las ocho líneas de menor peso. Y lo contrario, si alguna de las entradas de más peso se activara, la correspondiente línea $\overline{EO}$ se pondría a 1 y con ella la $\overline{EI}$, deshabilitando el codificador con las entradas de menos peso.

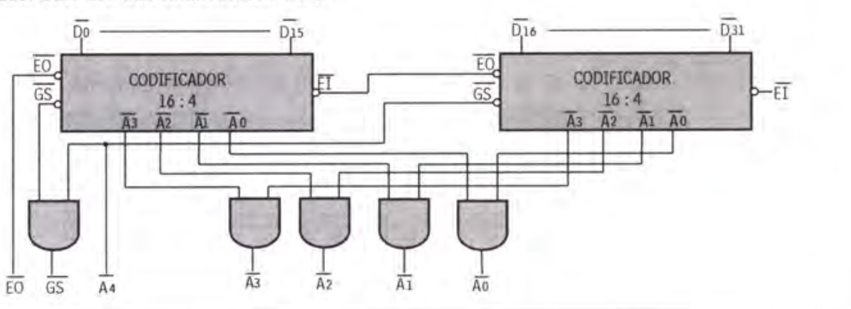
Ejemplo 4-3

Diseñar un codificador 32:5 con los codificadores 16:4 de la figura 4-9.

La estructura de la figura 4-9 se repite para un codificador 32:5. En este caso habría que sustituir cada 16:4 de la figura 4-10 por la estructura de la figura 4-9, resultando un circuito de carácter recursivo, lo que es típico de la extensión de sistemas combinacionales funcionales.

Figura 4-10

Codificador 32:5 con dos codificadores 16:4.



4.4.4. Funciones típicas de un codificador

Las funciones más usuales de un codificador son:

- Codificar en binario las teclas activadas en un teclado.
- Codificar distintas situaciones de la entrada bajo un mismo código.
- Codificar las señales obtenidas en conversores analógicos/digitales de código GRAY.

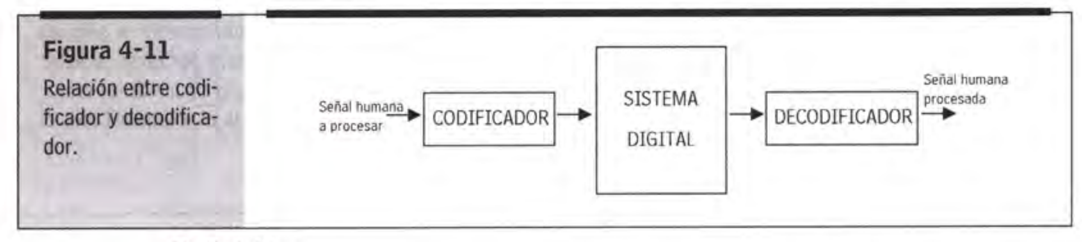
4.5. Decodificador

Un decodificador realiza la operación inversa al codificador, transforma una señal que expresa un contenido en código binario a otra que lo hace en código humano, del tipo 1 entre n.

La figura 4.11 muestra el caso típico en que la primera operación de un sistema digital es codificar lo humano en digital, y la última es decodificar lo digital en humano, y así permitir la interrelación entre sistemas con códigos distintos.

Figura 4-11

Relación entre codificador y decodificador.



Definición

Es un circuito con m entradas y n salidas, normalmente $n=2^m$. Cada combinación de valores de la entrada activa una y sólo una de las salidas; esta relación biunívoca se establece en base al código elegido, generalmente el binario puro o BCD, así hay decodificadores 3:8, 4:16 o 4:10; completos los dos primeros e incompleto el tercero.

También pueden diseñarse decodificadores donde para cada combinación de valores de la entrada se active más de una salida; éstos se denominan decodificadores-excitadores.

La tabla 4-6, las ecuaciones 4.6 y el esquema de la figura 4-12 describen un decodificador 3:8 con línea de ENABLE. La segunda tabla de verdad no lo es en sentido estricto pero es más compacta y explicativa.

Tabla 4-6

Tabla de verdad y tabla funcional del decodificador 3:8.

EI	A2	A1	A0	S0	S1	S2	S3	S4	S5	S6	S7
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

EI	A2	A1	A0	Si
0	X	X	X	0
1	0	0	0	S0
1	0	0	1	S1
1	0	1	0	S2
1	0	1	1	S3
1	1	0	0	S4
1	1	0	1	S5
1	1	1	0	S6
1	1	1	1	S7

En este caso no procede simplificar, y las ecuaciones que representan al decodificador son las propias formas normales.

$$
\begin{aligned}
S0 &= (\overline{A2} \cdot \overline{A1} \cdot \overline{A0}) \cdot EI & S1 &= (\overline{A2} \cdot \overline{A1} \cdot A0) \cdot EI \\
S2 &= (\overline{A2} \cdot A1 \cdot \overline{A0}) \cdot EI & S3 &= (A2 \cdot \overline{A1} \cdot \overline{A0}) \cdot EI \\
S4 &= (A2 \cdot \overline{A1} \cdot \overline{A0}) \cdot EI & S5 &= (A2 \cdot \overline{A1} \cdot A0) \cdot EI \\
S6 &= (A2 \cdot A1 \cdot \overline{A0}) \cdot EI & S7 &= (A2 \cdot A1 \cdot A0) \cdot EI
\end{aligned}
\qquad (4.6)
$$

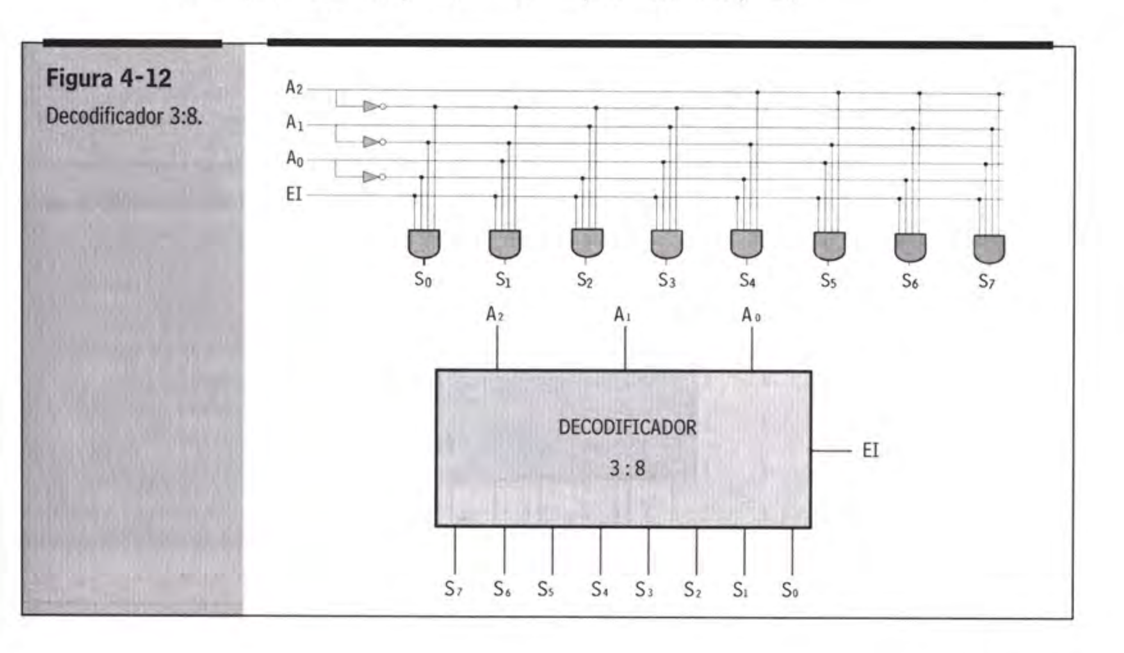


Figura 4-12

Decodificador 3:8.

En el caso de un decodificador 10:4, las seis combinaciones que van de 1010 a 1111 en la entrada pueden ser tratadas como condiciones libres (ver tabla 4.7), y así, mediante la simplificación, implementar un circuito con menor coste. De este modo, las ecuaciones correspondientes son las mostradas en 4.7.

$$S0 = \overline{A3} \cdot \overline{A2} \cdot \overline{A1} \cdot \overline{A0} \qquad (4.7)$$

$$S1 = \overline{A3} \cdot \overline{A2} \cdot \overline{A1} \cdot A0$$

$$S2 = \overline{A2} \cdot A1 \cdot \overline{A0} \qquad S3 = \overline{A2} \cdot A1 \cdot A0$$

$$S4 = A2 \cdot \overline{A1} \cdot \overline{A0} \qquad S5 = A3 \cdot \overline{A2} \cdot A1$$

$$S6 = A2 \cdot A1 \cdot \overline{A0} \qquad S7 = A2 \cdot A1 \cdot A0$$

$$S8 = A3 \cdot \overline{A0} \qquad S9 = A3 \cdot A0$$

Tabla 4-7
T-V decodificador 4:10.

A3	A2	A1	A0	Si	A3	A2	A1	A0	Si
0	0	0	0	S0	1	0	0	0	S8
0	0	0	1	S1	1	0	0	1	S9
0	0	1	0	S2	1	0	1	0	X
0	0	1	1	S3	1	0	1	1	X
0	1	0	0	S4	1	1	0	0	X
0	1	0	1	S5	1	1	0	1	X
0	1	1	0	S6	1	1	1	0	X
0	1	1	1	S7	1	1	1	1	X
0	0	1	1	S3	1	0	1	1	X
0	1	0	0	S4	1	1	0	0	X
0	1	0	1	S5	1	1	0	1	X
0	1	1	0	S6	1	1	1	0	X
0	1	1	1	S7	1	1	1	1	X

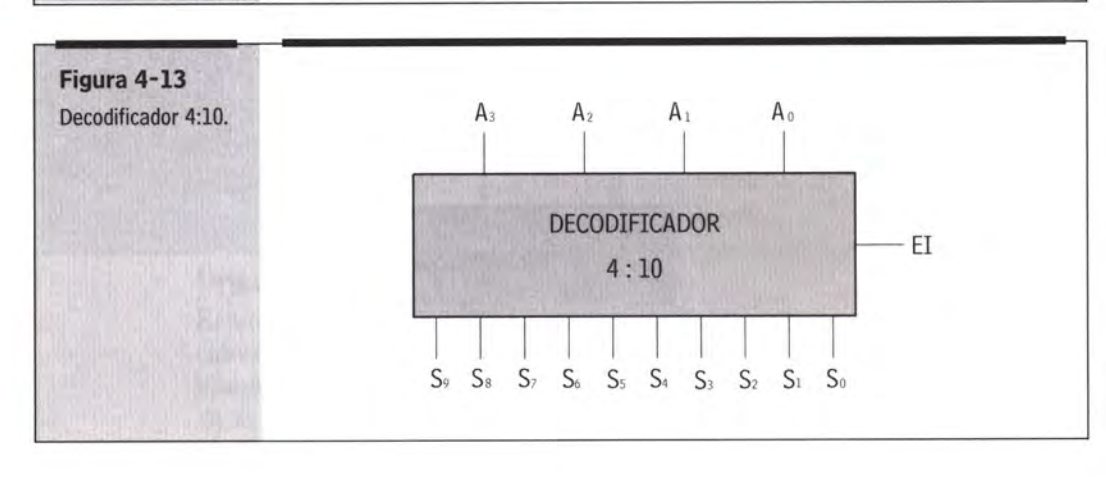


Figura 4-13
Decodificador 4:10.

En cuanto a la implementación en circuitos MSI, los decodificadores suelen tener las entradas a nivel alto y las salidas a nivel bajo, además suelen disponer de más de una entrada ENABLE para permitir una cómoda extensión de su capacidad.

Por ejemplo, el decodificador 3:8 74138 de la figura 4-14 es activo por nivel bajo en la salida y tiene tres señales ENABLE, dos activas a nivel bajo y una a nivel alto. La tabla de verdad es la tabla 4-8.

Figura 4-14

Decodificador 3:8 74138.

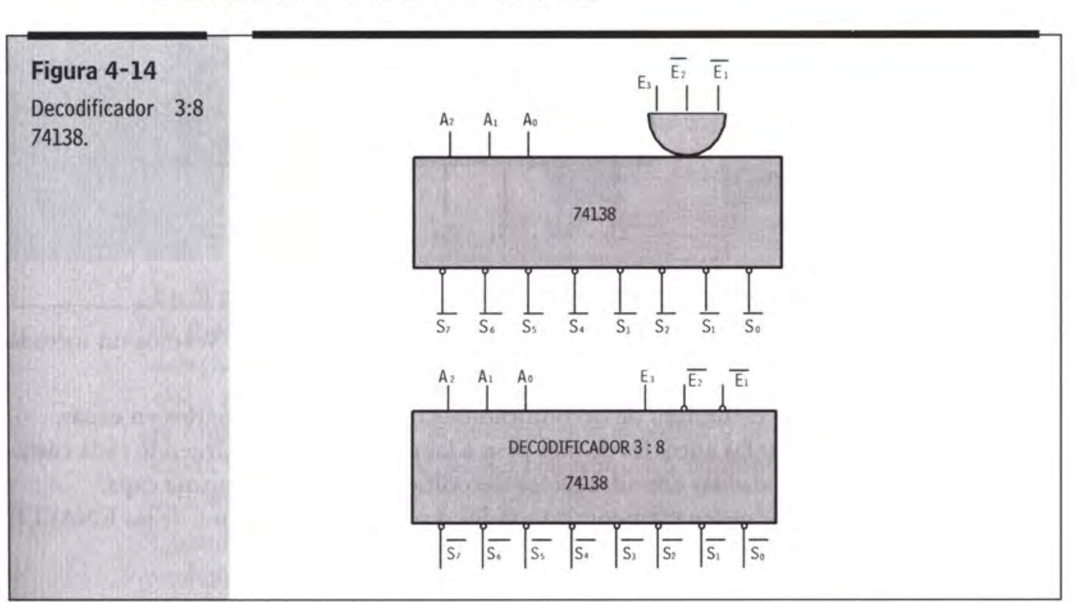


Tabla 4-8

Tabla de verdad del decodificador 74138.

E3	$\overline{E2}$	$\overline{E1}$	A2	A1	A0	$\overline{S0}$	$\overline{S1}$	$\overline{S2}$	$\overline{S3}$	$\overline{S4}$	$\overline{S5}$	$\overline{S6}$	$\overline{S7}$
0	X	X	X	X	X	1	1	1	1	1	1	1	1
1	1	X	X	X	X	1	1	1	1	1	1	1	1
1	X	1	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

Otro circuito decodificador es el doble decodificador 2:4 de la figura 4-15. Este MSI contiene dos decodificadores 2:4 independientes integrados en un único chip, el 74139.

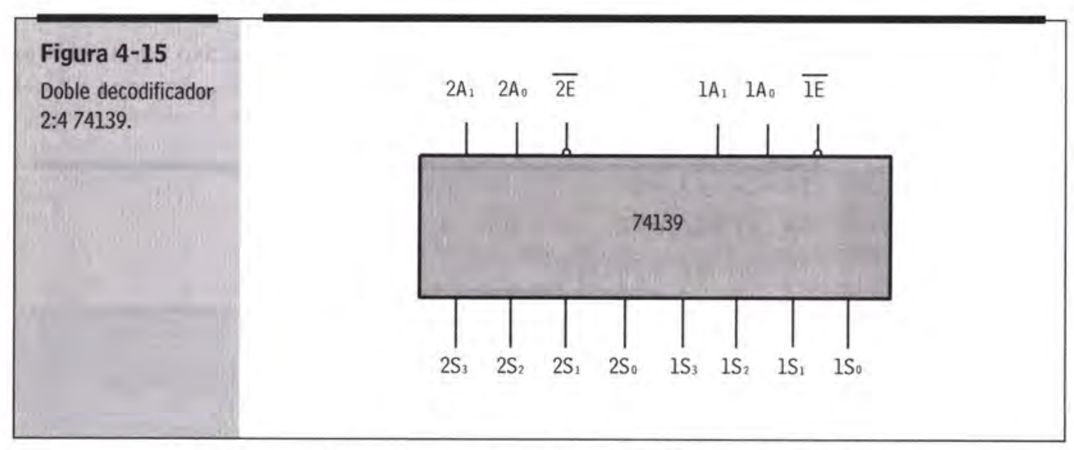


Figura 4-15

Doble decodificador 2:4 74139.

La extensión de un decodificador utiliza la línea ENABLE. Veamos un método ordenado de extensión de decodificadores:

- Elegir el número de decodificadores necesarios y ordenarlos en capas.
- Asociar las entradas de más peso a las capas más altas. Uniendo cada entrada a todas las entradas de los decodificadores de una misma capa.
- Asociar ordenadamente las salidas de una capa a cada una de las ENABLE de la capa inferior.
- Las salidas del decodificador están en la última capa.

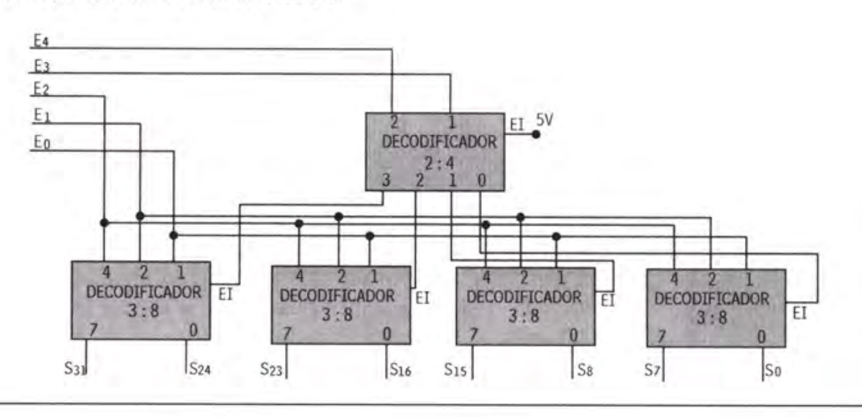
Ejemplo 4-4

Diseñar un decodificador 5:32 con decodificadores 3:8.

Al diseñar un 5:32 con decodificadores 3:8 y de 2:4 resulta la figura 4-16. En este caso bastará con dos *capas.*

Figura 4-16

Decodificador 5:32 con decodificadores 3:8.



Por otro lado, al diseñar un decodificador 3:8 con decodificadores 1:2 obtenemos la figura 4-17, con tres capas.

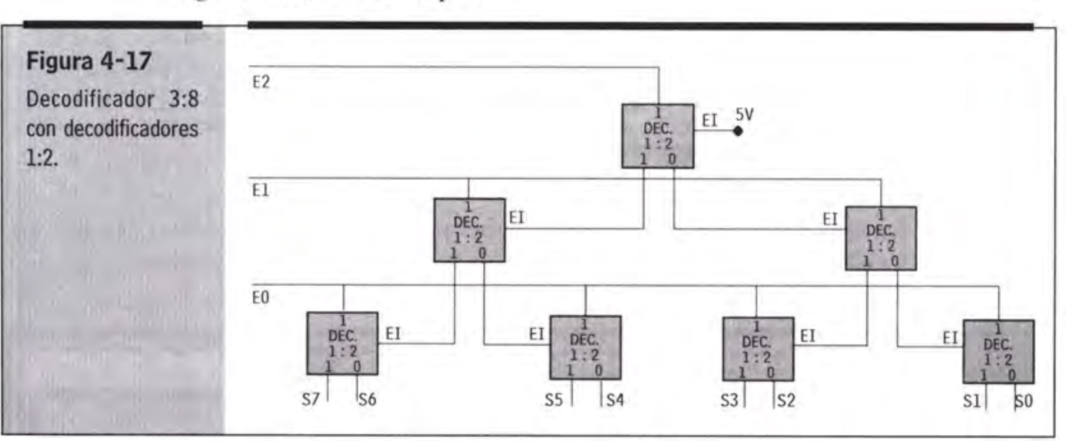


Figura 4-17
Decodificador 3:8 con decodificadores 1:2.

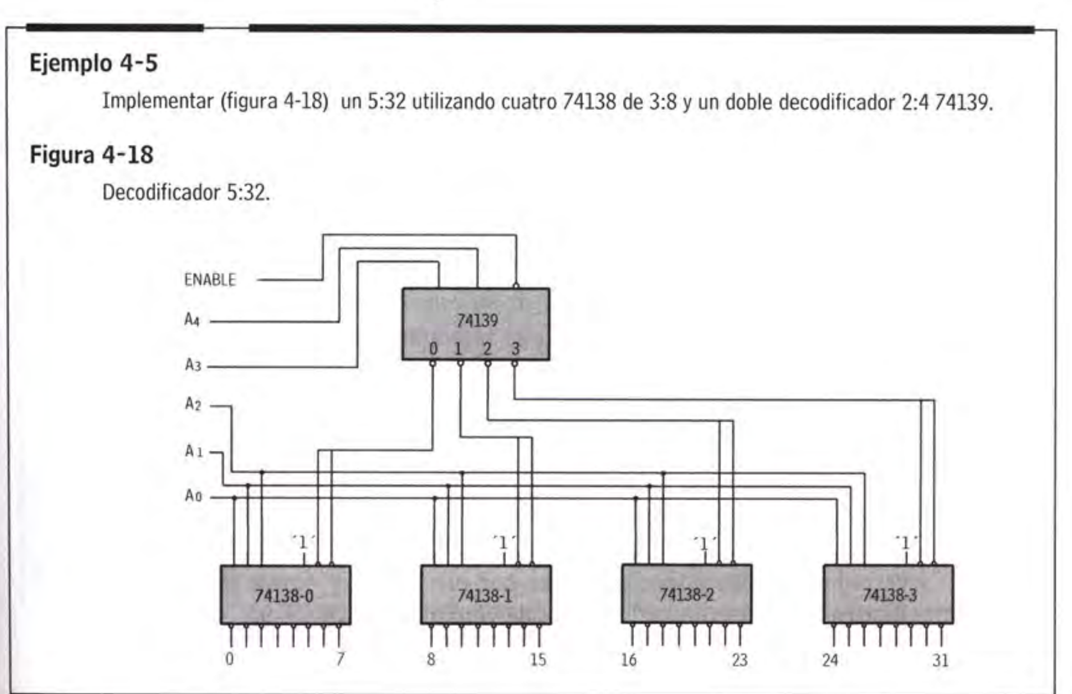
En el esquema de la figura 4-17 si la entrada fuera $E_{2-0}=010$ se activaría la salida 0 del primer decodificador ($E_2=0$), quedando activo el decodificador de la derecha de la segunda capa. Éste, al recibir 1 en su entrada activará su salida 1, que a su vez activará el ENABLE INPUT de un solo decodificador de la tercera capa, el segundo por la derecha, que al recibir 0 por sus entradas activará la salida 0, que es la salida S2 del conjunto. En conjunto vemos que con 010 se activará S2, como debía.

Cuando se utilizan circuitos MSI la extensión se parece a la anterior, pero aprovechando al máximo las posibilidades del CI.

Ejemplo 4-5

Implementar (figura 4-18) un 5:32 utilizando cuatro 74138 de 3:8 y un doble decodificador 2:4 74139.

Figura 4-18
Decodificador 5:32.



Ejemplo 4-5 (Continuación)

El planteamiento se basa en subdividir la salida en bloques de ocho, así habrá cuatro bloques. De los cuatro posibles sólo uno de ellos se activará según el valor de las dos líneas de más peso en la entrada: si A4A3 = 00 se refiere a una de las ocho salidas de menor valor, mientras A4A3 = 10 se refiere a las salidas $\overline{S16}$ a $\overline{S23}$, y así para el resto de valores de A4A3.

Por ejemplo A4A3A2A1A0 = 10110, activa la séptima salida del bloque 2 denominada $\overline{S22}$, no quedando activada ninguna de las restantes.

En el anterior circuito podríamos haber prescindido del decodificador doble 2:4 y utilizar la puerta AND asociada a las líneas ENABLE. El circuito de la figura 4-19 pierde claridad pero es más económico.

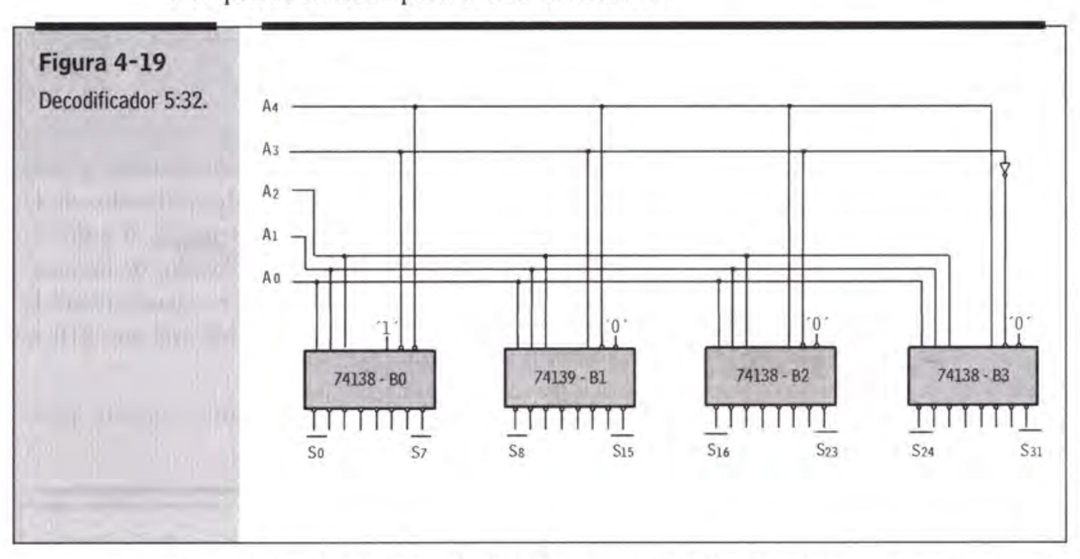


Figura 4-19
Decodificador 5:32.

Por último, el decodificador 74154 es un decodificador 4:16 con la entrada activa por nivel alto y la salida por nivel bajo y dos líneas de ENABLE, ambas activas por nivel bajo.

4.5.1. Decodificador-excitador BCD 7 segmentos

En este caso, cada combinación de valores en la entrada no activa una salida, sino varias. En general, estas señales activarán a su vez dispositivos de tipo luminiscentes en forma de barra -diodos led- que visualizarán los símbolos correspondientes a cada combinación de valores de la entrada. En la figura 4-20 vemos la disposición de los siete segmentos.

Los decodificadores/excitadores dependen mucho de la disposición de diodos elegida, de todos ellos el más utilizado es el 7 segmentos. El circuito integrado 7447 implementa un decodificador BCD/7 segmentos donde la entrada es activa por nivel alto y la salida lo es por nivel bajo, y donde las señales $\overline{LT}$, $\overline{RBI}$, $\overline{BI/RBO}$ son de control. La tabla de verdad es la tabla 4-9.

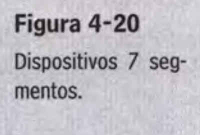


Figura 4-20

Dispositivos 7 segmentos.

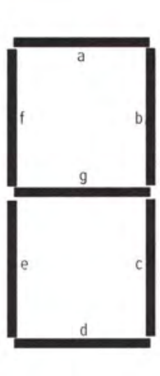


Tabla 4-9

Tabla de verdad del BCD 7 segmentos 7447.

Entradas							Salidas						
$\overline{LT}$	$\overline{RBI}$	E3	E2	E1	E0	$\overline{BI}$/ $\overline{RBO}$	$\overline{a}$	$\overline{b}$	$\overline{c}$	$\overline{d}$	$\overline{e}$	$\overline{f}$	$\overline{g}$
X	X	X	X	X	X	0	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
0	X	X	X	X	X	1	0	0	0	0	0	0	0
1	1	0	0	0	0	1	0	0	0	0	0	0	1
1	X	0	0	0	1	1	1	0	0	1	1	1	1
1	X	0	0	1	0	1	0	0	1	0	0	1	0
1	X	0	0	1	1	1	0	0	0	0	1	1	0
1	X	0	1	0	0	1	1	0	0	1	1	0	0
1	X	0	1	0	1	1	0	1	0	0	1	0	0
1	X	0	1	1	0	1	1	1	0	0	0	0	0
1	X	0	1	1	1	1	0	0	0	1	1	1	1
1	X	1	0	0	0	1	0	0	0	0	0	0	0
1	X	1	0	0	1	1	0	0	0	1	1	0	0
1	X	1	0	1	0	1	1	1	1	0	0	1	0
1	X	1	0	1	1	1	1	1	0	0	1	1	0
1	X	1	1	0	0	1	1	0	1	1	1	0	0
1	X	1	1	0	1	1	0	1	1	0	1	0	0
1	X	1	1	1	0	1	1	1	1	0	0	0	0
1	X	1	1	1	1	1	1	1	1	1	1	1	1

Desde el punto de vista electrónico los 7 segmentos pueden ser de dos tipos:

- Cátodo común. El led se enciende (se activa) si es excitado en su entrada con Vcc (5 V). En este caso los 7 diodos led tienen el cátodo en común, que en principio ha de estar conectado a tierra.
- Ánodo común. El led se enciende (se activa) si es excitado en su entrada con tierra (0 V). En este caso los 7 diodos led tienen el ánodo en común, que en principio ha de estar conectado a Vcc.

La figura 4-21 visualiza un 3 en dos 7 segmentos; el de la izquierda de cátodo común y el de la derecha de ánodo común.

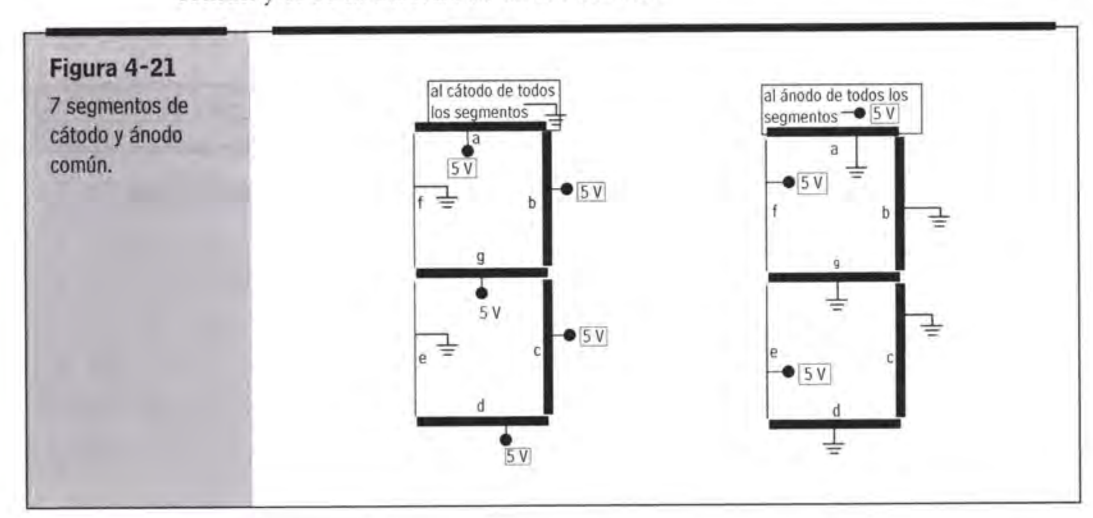


Figura 4-21
7 segmentos de cátodo y ánodo común.

Las líneas de control representadas en la tabla 4-9 influyen de la siguiente manera:

- $\overline{LT}$: si se activa la línea -por nivel bajo- todos los segmentos se activan para comprobar su buen funcionamiento. LAMP TEST.
- $\overline{RBI}$: sólo afecta al decodificador si la entrada es 0000, -0 decimal-. Si $\overline{RBI}$ está activa el 0 no se visualiza; en caso contrario, si $\overline{RBI}=1$, sí se visualiza.
- $\overline{BI/RBO}$: esta línea puede comportarse como entrada o salida. Como línea de entrada, si se activa $\overline{BI}$ -nivel bajo- todos los segmentos se desactivan, sea cual sea la entrada. Como salida $\overline{RBO}$ se activa -a nivel bajo- cuando estando $\overline{RBI}=0$, la entrada es 0000; esta línea se utiliza para la conexión en cascada. Se puede ver que BI y RBO son coherentes entre sí, aun siendo entrada y salida.

La utilización de las líneas $\overline{RBI}$ y $\overline{RBO}$ permite que no se visualicen los ceros a la izquierda del dígito más significativo. Observando la figura 4-22 vemos que si la entrada decimal fuera 001 el primer cero no se visualizaría por serlo y estar $\overline{RBI}$ a tierra; esta situación hace que el primer 7447 entregue un nivel bajo por $\overline{RBO}$ que llegará al segundo circuito integrado a través de $\overline{RBI}$, y por tanto tampoco se visualizará el segundo 0. El 0 entregado por $\overline{RBO}$ no tiene efecto en el tercer circuito integrado, ya que el dígito a visualizar es el 1; además, aunque fuera un 0, éste se visualizaría por ser el último y estar su $\overline{RBI}$ a 5 V.

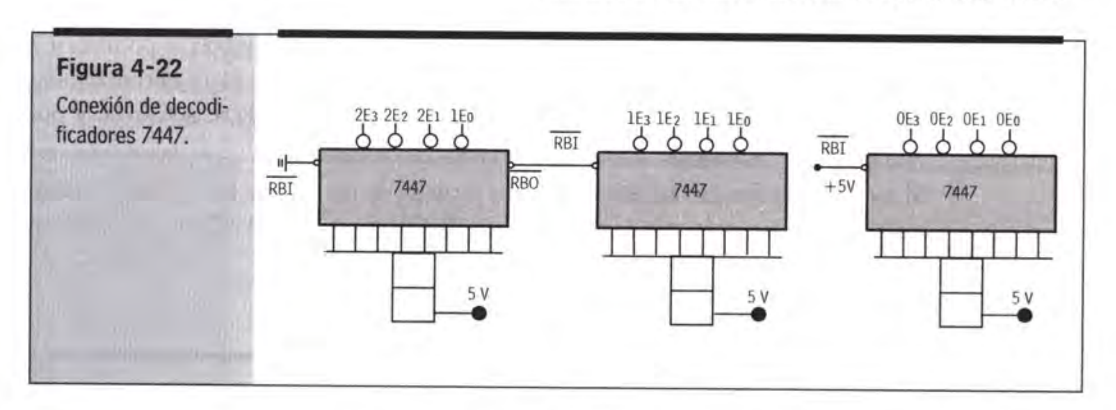


Figura 4-22
Conexión de decodificadores 7447.

Las salidas de los 7447 de la figura 4-22 son activas por nivel bajo, y por tanto el 7 segmentos ha de ser de ánodo común, alimentado con 5 voltios. Mientras que en el 7448 al ser las salidas activas por nivel alto, el 7 segmentos debe ser de cátodo común, alimentado con tierra.

4.5.2. Implementación de funciones booleanas con decodificadores

Mediante un decodificador n:2^n y una puerta OR, se puede implementar cualquier función booleana de n variables descrita por su forma normal disyuntiva.

Como muestra el ejemplo de la figura 4-23, el procedimiento consiste simplemente en reunir mediante la suma lógica las salidas del decodificador que pertenezcan a la forma normal disyuntiva.

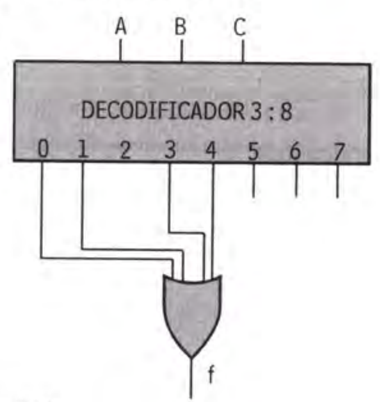
Ejemplo 4-6

Implementar f con un decodificador.

$$\sum(0, 1, 3, 4) = m_0 + m_1 + m_3 + m_4$$

Figura 4-23

Implementación de una función de tres variables con un 3:8.



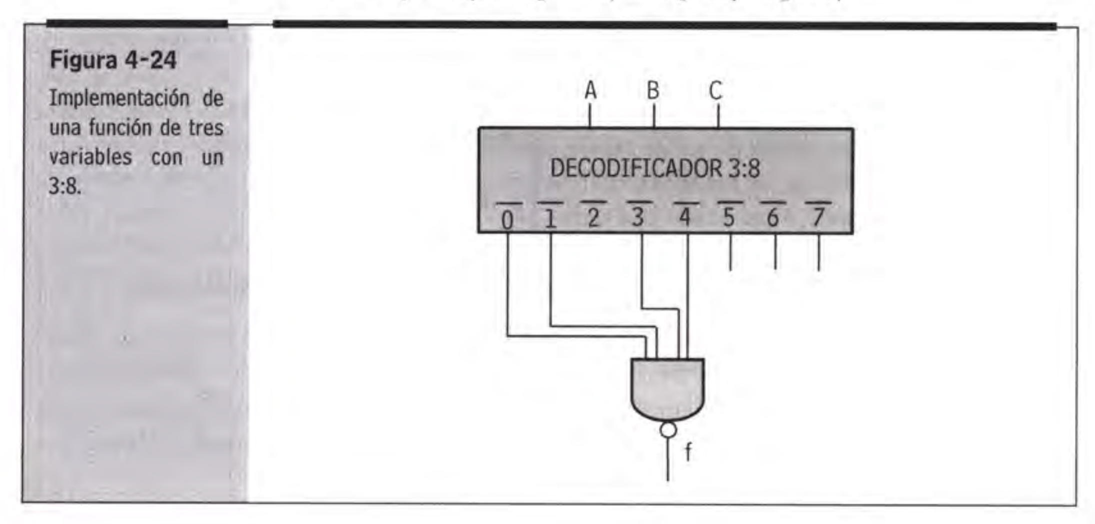
La función f ha quedado implementada.

Por ejemplo, si en el anterior circuito la entrada fuera ABC=000 se pondría a 1 la salida 0 y con ella la salida f a través de la puerta OR. Si la entrada fuera ABD=111 ninguna de las cuatro entradas de la puerta OR se activaría, y por tanto f quedaría a cero.

Si en el decodificador las salidas fueran activas por nivel bajo, la implementación sería posible sin más que cambiar la puerta OR por una NAND, como se ve en la figura 4-24, aplicando el teorema de Morgan.

$$f = m_0 + m_1 + m_3 + m_4 = \overline{\overline{m_0} \cdot \overline{m_1} \cdot \overline{m_3} \cdot \overline{m_4}}$$

Figura 4-24
Implementación de una función de tres variables con un 3:8.



Si la función booleana a implementar no estuviera expresada por su forma normal disyuntiva, la implementación pasaría por manipularla booleanamente hasta obtener su forma normal.

Aplicaciones típicas con decodificadores son:

- Visualización de resultados.
- Direccionamiento en una memoria RAM o ROM.
- Direccionamiento de los distintos dispositivos de una CPU.
- Implementación de funciones booleanas.
- Implementación de demultiplexores.

4.6. Multiplexores

Un multiplexor tiene varias entradas disponibles, de todas ellas una y sólo una pasa a la salida, es decir, el multiplexor ofrece en la salida el contenido sin modificar de la entrada seleccionada. Podemos observar un multiplexor como un distribuidor de las entradas hacia la única salida, o, nombrando a algunas traducciones, como un *encaminador*.

Definición

Un multiplexor 2^n:1 es un dispositivo con 2^n entradas y una salida. El contenido de una de las entradas pasa a la salida según el valor de las n líneas de control.

La figura 4-25 muestra el multiplexor como un conmutador con varias posiciones posibles que conecta, encamina o distribuye en cada caso una de las entradas -siempre una y sólo una- a la salida.

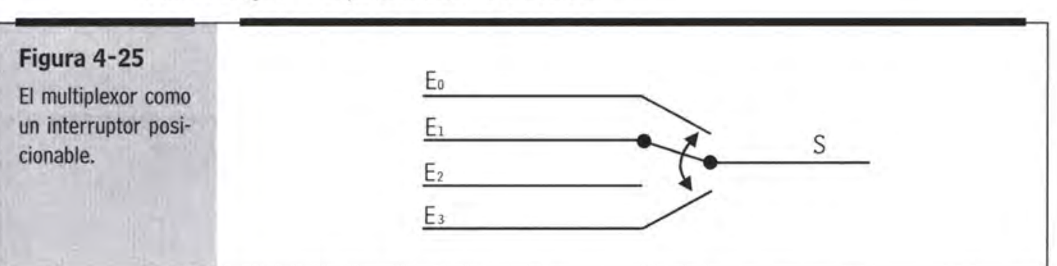


Figura 4-25
El multiplexor como un interruptor posicionable.

En el esquema 4-25 y en las tablas de verdad -tabla 4-10- observamos que la entrada que se conecta a la salida es aquella cuyo valor decimal coincide con el valor binario de las líneas de control. Por ejemplo, para conectar la entrada E2 a la salida las líneas de control deben ser C1C0=10.

Tabla 4-10
Tabla de verdad y tabla funcional de un multiplexor 4:1.

E3	E2	E1	E0	C1	C0	S
X	X	X	0	0	0	0
X	X	X	1	0	0	1
X	X	0	X	0	1	0
X	X	1	X	0	1	1
X	0	X	X	1	0	0
X	1	X	X	1	0	1
0	X	X	X	1	1	0
1	X	X	X	1	1	1

C1	C0	S
0	0	E0
0	1	E1
1	0	E2
1	1	E3

La ecuación booleana 4.8 se obtiene directamente de la funcionalidad del multiplexor. El lector debe observar en la expresión la presencia e importancia de la decodificación.

$$S = E0\overline{C1}\,\overline{C0} + E1\overline{C1}C0 + E2C1\overline{C0} + E3C1C0 \qquad (4.8)$$

El circuito correspondiente a la ecuación 4.8 es el de la figura 4-26.

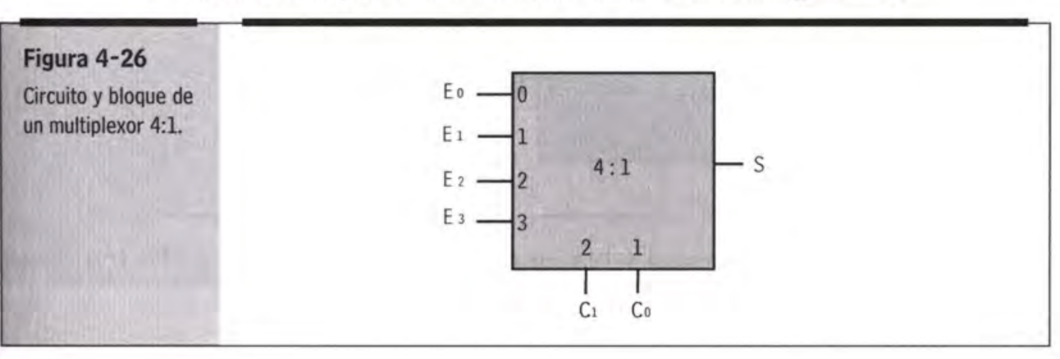


Figura 4-26
Circuito y bloque de un multiplexor 4:1.

Al igual que en otros elementos los multiplexores poseen una línea ENABLE -generalmente activa por nivel bajo- que si no estuviera activa haría que la salida tomara el valor cero, independientemente de las líneas de entrada y control.

El multiplexor MX 8:1 74151 descrito en la figura 4-27 y en la tabla 4-11 posee línea de ENABLE y la salida aparece duplicada como S y $\overline{S}$.

Tabla 4-11

Tabla funcional del Mx 8:1 74151.

$\overline{EI}$	C2	C1	C0	S	$\overline{S}$
1	X	X	X	0	1
0	0	0	0	E0	$\overline{E0}$
0	0	0	1	E1	$\overline{E1}$
0	0	1	0	E2	$\overline{E2}$
0	0	1	1	E3	$\overline{E3}$
0	1	0	0	E4	$\overline{E4}$
0	1	0	1	E5	$\overline{E5}$
0	1	1	0	E6	$\overline{E6}$
0	1	1	1	E7	$\overline{E7}$

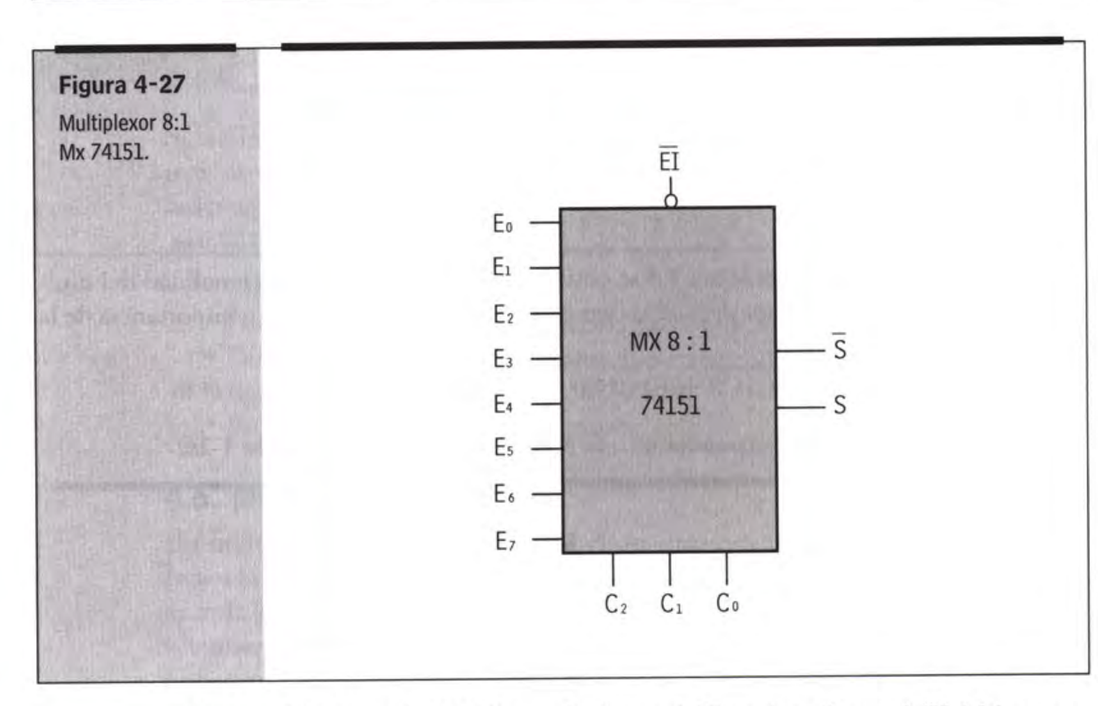


Figura 4-27

Multiplexor 8:1 Mx 74151.

En la tabla de verdad 4-12 y en la figura 4-28 se describe un MX 16:1 correspondiente al circuito integrado 74150.

Tabla 4-12
Tabla de un Mx 16:1 74150.

$\overline{EI}$	C3	C2	C1	C0	S	$\overline{S}$
1	X	X	X	X	0	1
0	0	0	0	0	E0	$\overline{E0}$
0	0	0	0	1	E1	$\overline{E1}$
0	0	0	1	0	E2	$\overline{E2}$
0	0	0	1	1	E3	$\overline{E3}$
0	0	1	0	0	E4	$\overline{E4}$
0	0	1	0	1	E5	$\overline{E5}$
0	0	1	1	0	E6	$\overline{E6}$
0	0	1	1	1	E7	$\overline{E7}$
0	1	0	0	0	E8	$\overline{E8}$
0	1	0	0	1	E9	$\overline{E9}$
0	1	0	1	0	E10	$\overline{E10}$
0	1	0	1	1	E11	$\overline{E11}$
0	1	1	0	0	E12	$\overline{E12}$
0	1	1	0	1	E13	$\overline{E13}$
0	1	1	1	0	E14	$\overline{E14}$
0	1	1	1	1	E15	$\overline{E15}$

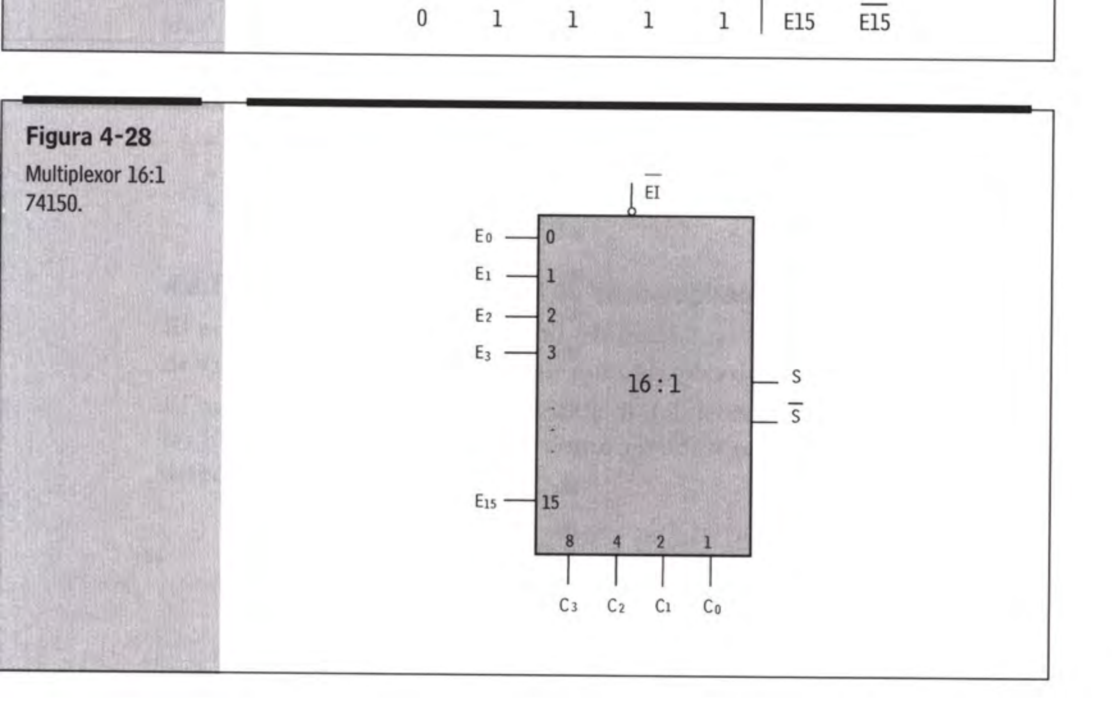


Figura 4-28
Multiplexor 16:1 74150.

Tecnológicamente se implementan los multiplexores 2:1, 4:1, 8:1 y 16:1; también son muy útiles otros dos tipos de multiplexores:

- Cuatro multiplexores 2:1 en un mismo circuito integrado -74157-, compartiendo la línea de control.
- Dos multiplexores 4:1 en un mismo circuito integrado -74153-, compartiendo las líneas de control.

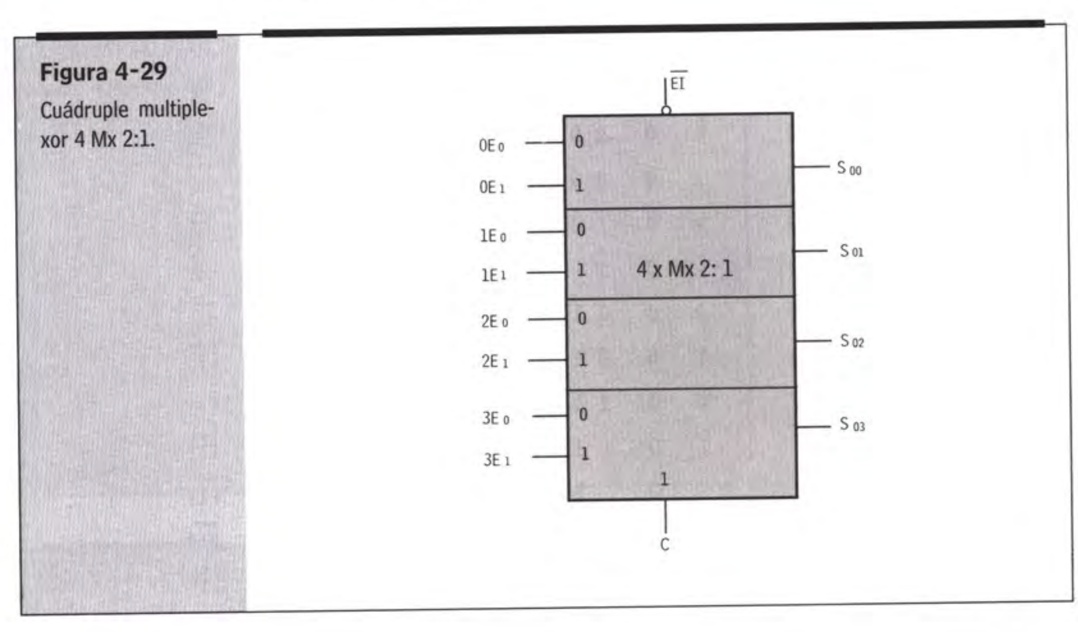


Figura 4-29
Cuádruple multiplexor 4 Mx 2:1.

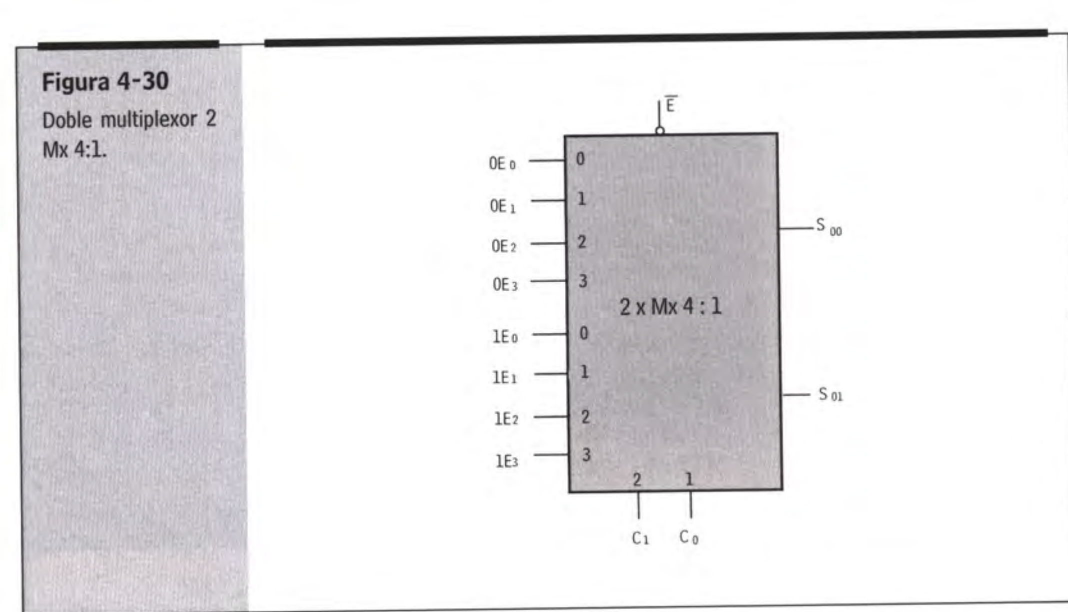


Figura 4-30
Doble multiplexor 2 Mx 4:1.

BIBLIOTECA CADIZ

Tabla 4-13
Tabla del cuádruple multiplexor 4xMx2:1.

E	C	S0	S1	S2	S3
1	X	0	0	0	0
0	0	0E0	1E0	2E0	3E0
0	1	0E1	1E1	2E1	3E1

Tabla 4-14
Tabla de un doble multiplexor 2 Mx 4:1.

$\overline{E}$	C1	C0	$\overline{S0}$	S0	S1	$\overline{S1}$
1	X	X	0	1	0	1
0	0	0	0E0	$\overline{0E0}$	1E0	$\overline{1E0}$
0	0	1	0E1	$\overline{0E1}$	1E1	$\overline{1E1}$
0	1	0	0E2	$\overline{0E2}$	1E2	$\overline{1E2}$
0	1	1	0E3	$\overline{0E3}$	1E3	$\overline{1E3}$

En ambos casos, como muestran las figuras 4-29 y 4-30 y las tablas 4-13 y 4-14, las líneas de control son comunes a todos los multiplexores del circuito integrado y disponen de línea ENABLE. Así, si C0=1 se conecta la entrada 1 de cada Mx a la correspondiente salida, es decir, no se puede conectar en un Mx la entrada 0, en otro la 1, etc.

Las funciones más típicas de un multiplexor son:

- Distribuidor de señales.
- En la implementación de ALU's.
- Para el uso compartido de buses de datos.
- Implementación de funciones booleanas.

4.6.1. Extensión de la capacidad de un multiplexor

El multiplexor de mayor capacidad fabricado es el MX 16:1, pero la asociación de varios de éstos permite construir multiplexores de mayor capacidad.

El método a aplicar es muy parecido al del decodificador: varias capas con las líneas de control ordenadas de un menor a mayor peso, de izquierda a derecha.

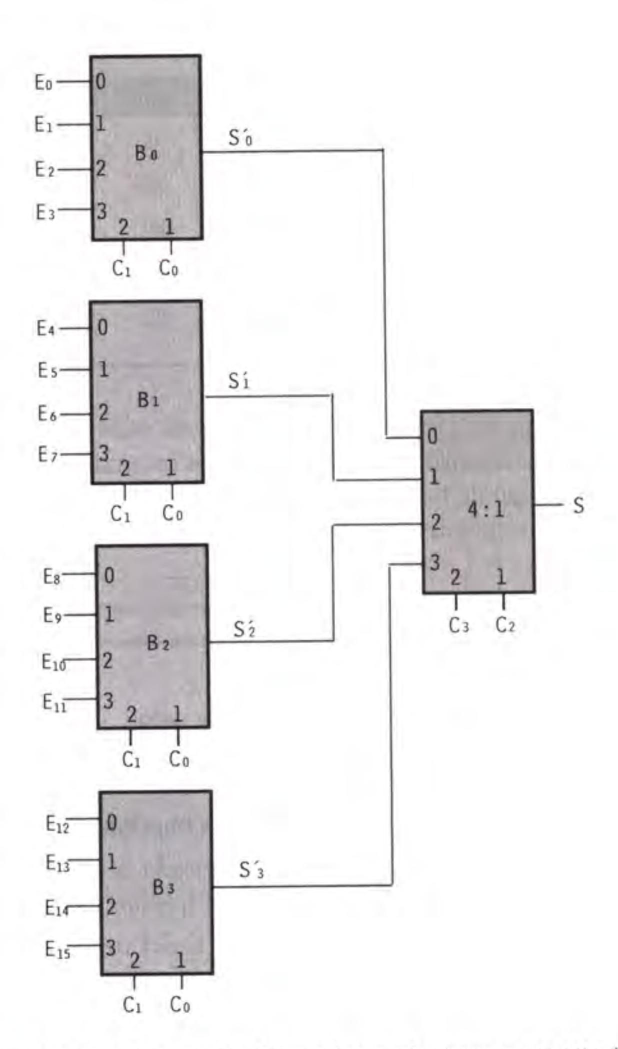
Ejemplo 4-7

Diseñar un multiplexor 16:1 con multiplexores 4:1.

Con cinco multiplexores 4:1 se puede implementar un 16:1, como se puede ver en la figura 4-31.

Figura 4-31

Multiplexor 16:1 con cinco Mx 4:1.



En este caso la multiplexión se hace en dos niveles, primero seleccionamos una entrada de cada bloque, y en el último 4:1 seleccionamos un bloque. Por ejemplo, si C3C2C1C0=0111, en el primer nivel tenemos seleccionados a E3, E7, E11 y E15, y es en el último nivel cuando entre estos cuatro seleccionamos el correspondiente a la línea E7 (C3C2 = 01).

Ejemplo 4-8

Implementar un multiplexor 16:1 con 15 multiplexores 2:1, como se puede observar en la figura 4-32.

Figura 4-32

Multiplexor 16:1 con quince Mx 2:1.

Vemos que extender la capacidad de un multiplexor pasa por dividir el multiplexado en fases, cada una de ellas con sus correspondientes líneas de control. En principio lo mejor es asignar las líneas de control de menos peso a las primeras fases, reservando las de más peso para la última fase.

4.6.2. Implementación de funciones booleanas con multiplexores

Cualquier función booleana puede ser implementada con un multiplexor. Por ejemplo, una función de cuatro variables se implementa con un multiplexor 8:1, y una de tres variables con un multiplexor 4:1.

Para implementar una función con un multiplexor:

- Una de las líneas de entrada de la función se va a asignar a las entradas del Mx.
- El resto de entradas se utiliza como líneas de control del multiplexor.
- La función debe ser manipulada hasta que en todos sus términos aparezcan todas las líneas de entrada asociadas a las líneas de control del Mx (punto 1).
- La función booleana f debe ser reescrita hasta su forma normal disyuntiva.
- A la vista de la forma normal, asociaremos cada entrada del Mx el valor correspondiente de la línea elegida en el primer punto.

Ejemplo 4-9

Implementar con un multiplexor $f = A \cdot B \cdot C + A \cdot \overline{B} \cdot C \cdot D + \overline{B} \cdot \overline{D}$

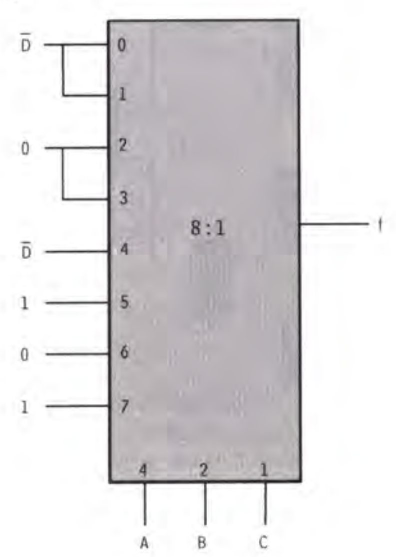
Asociamos D a la entrada del multiplexor y A, B y C a los controles.

Manipulamos la expresión hasta que todos los términos tengan A, B y C (elegidas como líneas de control del Mx). Multiplicamos cada uno por 0, 1, D o $\overline{D}$.

$$f = A \cdot B \cdot C \cdot 1 + A \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot C \cdot \overline{D} +$$
$$+ A \cdot B \cdot \overline{C} \cdot 0 + \overline{A} \cdot B \cdot \overline{C} \cdot 0 + \overline{A} \cdot B \cdot C \cdot 0 =$$

Figura 4-33

Implementación de f con un 8:1



Este circuito implementa la función f.

4.6.3. Multiplexores analógicos

Los multiplexores vistos en este epígrafe son todos digitales, pero también existen los analógicos. En estos dispositivos, tanto las señales de entrada como las de salida son analógicas.

4.7. Demultiplexores

Un demultiplexor tiene una entrada y varias salidas, de todas ellas una y sólo una se conecta a la salida, es decir, el demultiplexor ofrece en la salida seleccionada el contenido sin modificar de la entrada. Un demultiplexor es el dispositivo complementario del multiplexor.

Definición

Un demultiplexor 1: 2^n es un dispositivo con una entrada y 2^n salidas. El contenido de la entrada pasa a la salida seleccionada según el valor de las n líneas de control.

Las figuras 4-34 y 4-35 nos muestran el demultiplexor como un conmutador, y el multiplexor conectado con un demultiplexor para obtener un sistema de comunicación basado en una única línea de transmisión.

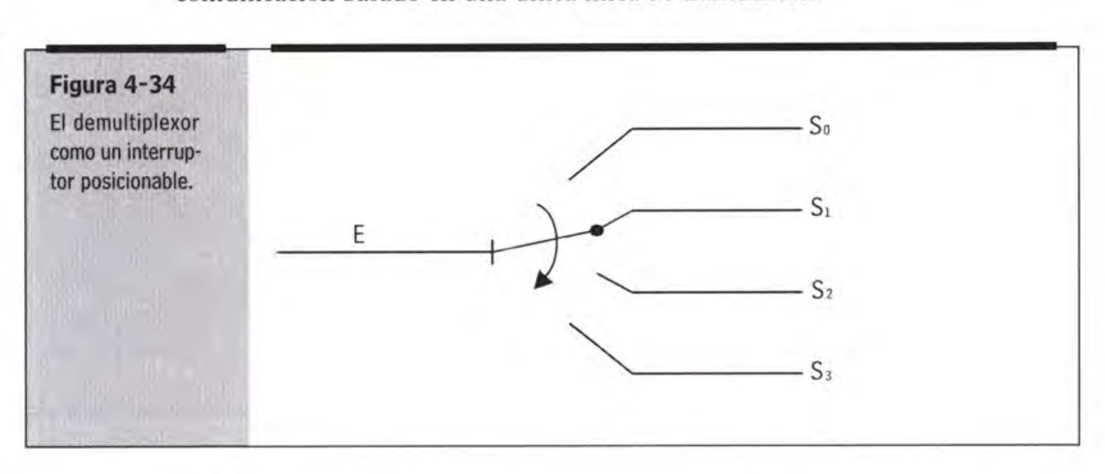


Figura 4-34
El demultiplexor como un interruptor posicionable.

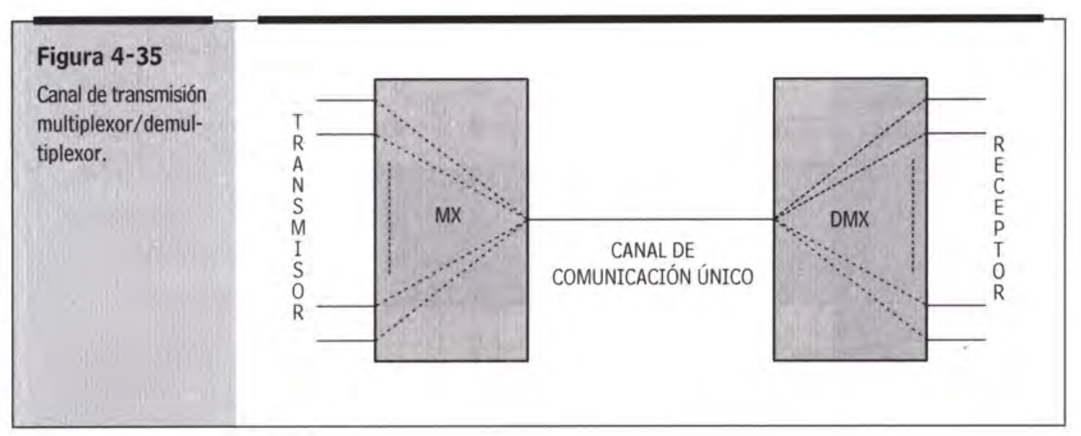


Figura 4-35
Canal de transmisión multiplexor/demultiplexor.

En el circuito lógico 4-36, la ecuación booleana 4.9 y la tabla de verdad 4-15 observamos que la salida que se conecta a la entrada es aquella cuyo valor decimal coincide con el valor binario de las líneas de control. Por ejemplo, para conectar la salida 2 a la entrada, las líneas de control deben ser C1C0=10.

Figura 4-36

Circuito y bloque de un demultiplexor 1:4.

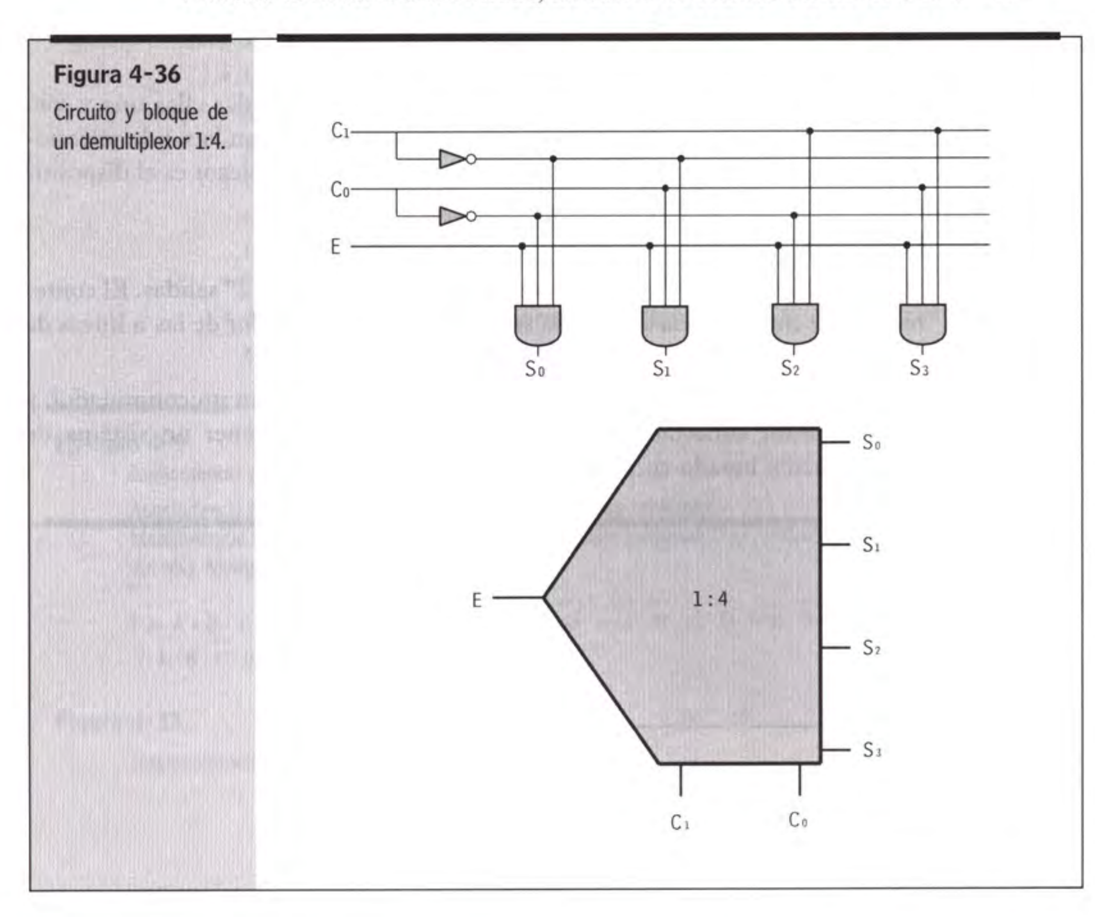


Tabla 4-15

Tabla de verdad y tabla funcional de un demultiplexor 1:4.

E	C1	C0	S3	S2	S1	S0
0	0	0	0	0	0	0
1	0	0	0	0	0	1
0	0	1	0	0	0	0
1	0	1	0	0	1	0
0	1	0	0	0	0	0
1	1	0	0	1	0	0
0	1	1	0	0	0	0
1	1	1	1	0	0	0

C1	C0	Sal
0	0	S0=E
0	1	S1=E
1	0	S2=E
1	1	S3=E

$$S0 = (\overline{A1} \cdot \overline{A0}) \cdot E \qquad S1 = (\overline{A1} \cdot A0) \cdot E$$

$$S2 = (A1 \cdot \overline{A0}) \cdot E \qquad S3 = (A1 \cdot A0) \cdot E \qquad (4.9)$$

En realidad los demultiplexores no son fabricados y vendidos como tales, sino que su funcionalidad se identifica con la de un decodificador cuyas líneas de entrada son las de control, y en el que la línea de entrada de datos del demultiplexor es conectada al ENABLE del decodificador. Así, los circuitos 74154 y 74138 son vendidos como decodificadores/demultiplexores. Por ejemplo, la figura 4-37 muestra la conexión de un decodificador 3:8 y la de un 4:16 para obtener sendos demultiplexores 1:8 y 1:16, respectivamente.

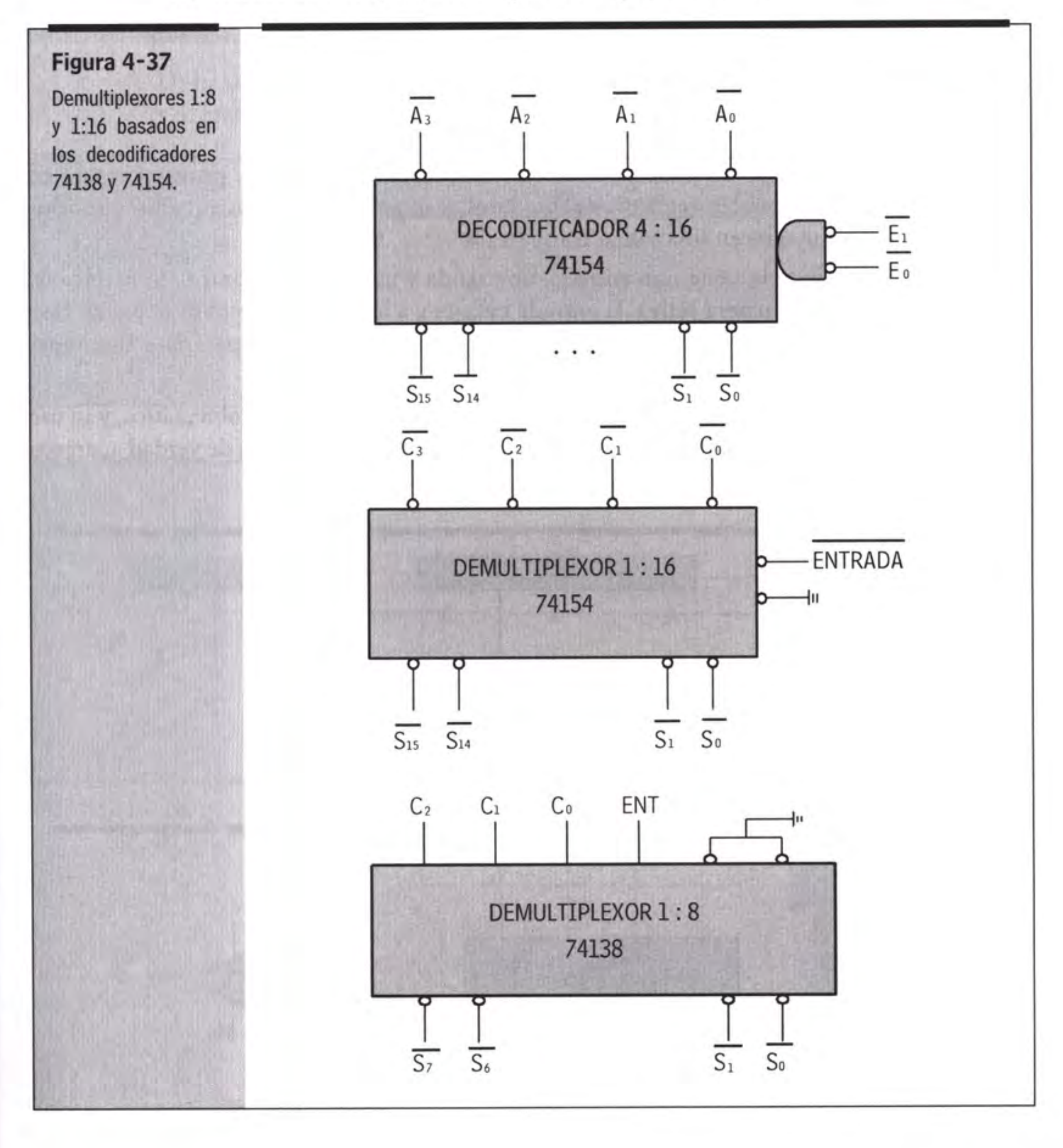


Figura 4-37
Demultiplexores 1:8 y 1:16 basados en los decodificadores 74138 y 74154.

En los circuitos anteriores, el valor que toman las salidas no conectadas a la entrada no es 0, sino 1, puesto que la salida del decodificador es por nivel bajo. Esta circunstancia ha de ser tenida en cuenta al usarlos.

A la hora de plantear la extensión de capacidad de un demultiplexor, ésta se realiza siguiendo las pautas de la extensión del decodificador, con la única anotación de que ahora disponemos de una línea menos de ENABLE para la extensión, ya que ésta da servicio a la entrada.

Otro tipo de demultiplexor es aquel cuyas salidas no conectadas no quedan a un valor preestablecido -ya sea éste 0 o 1-, sino que quedan en estado de alta impedancia.

Una línea en alta impedancia se comporta como un circuito abierto, es decir, *como si no existiera*. Cuando una línea o dispositivo está en alta impedancia no afecta en absoluto al resto del circuito.

4.7.1. Elemento triestado

El elemento triestado es utilizado en muchos dispositivos principalmente en demultiplexores, registros y contadores, y en general en todos aquellos dispositivos que ofrecen una salida compartida.

Un triestado tiene una entrada, una salida y una línea de control. Si la línea de control estuviera activa, la entrada pasaría a a la salida sin modificación; en caso contrario, si la línea de control estuviera inactiva, la salida pasaría a alta impedancia (Z), al tercer estado.

Como podemos imaginar este dispositivo es complicado y problemático, y su uso debe ser tan restringido como justificado. Su símbolo y tabla de verdad aparecen en la figura 4-38 y en la tabla 4-16, respectivamente.

Tabla 4-16
Tabla de verdad de un triestado.

E	C	S
0	1	0
1	1	1
0	0	T
1	0	T

C	S
1	E
0	Z

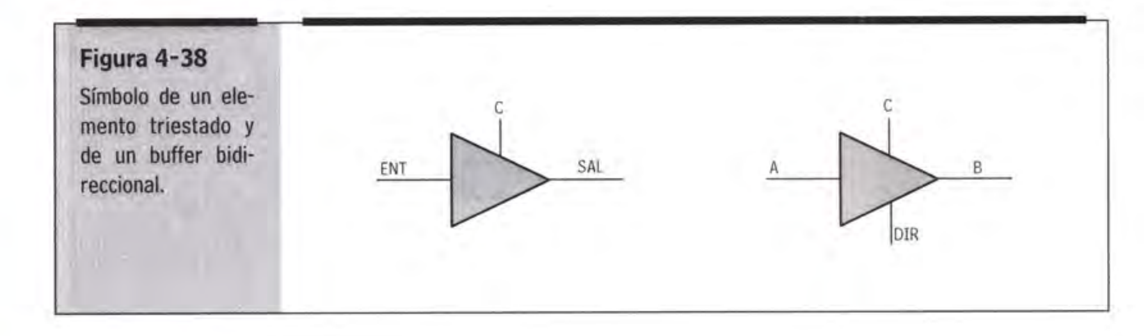


Figura 4-38
Símbolo de un elemento triestado y de un buffer bidireccional.

Cuando un triestado es utilizado en un circuito se le denomina buffer. Además, existe el buffer bidireccional (derecha de la figura 4-38) que permite elegir el sentido del flujo de la información de A a B o de B a A, según sea el valor de una entrada auxiliar DIR. La tabla 4-17 resume este comportamiento.

Tabla 4-17
T-V de un buffer bidireccional.

C	DIR	A	B	Función
0	X	Z	Z	Alta Impedancia
1	0	entrada	A	A → B
1	1	B	entrada	B → A

Utilizando triestados un demultiplexor 1:4 respondería a la tabla de verdad 4-18 y al circuito de la figura 4-39.

Tabla 4-18
T-V de un multiplexor 1:4 con triestado.

C1	C0	S0	S1	S2	S3
0	0	E	T	T	T
0	1	T	E	T	T
1	0	T	T	E	T
1	1	T	T	T	E

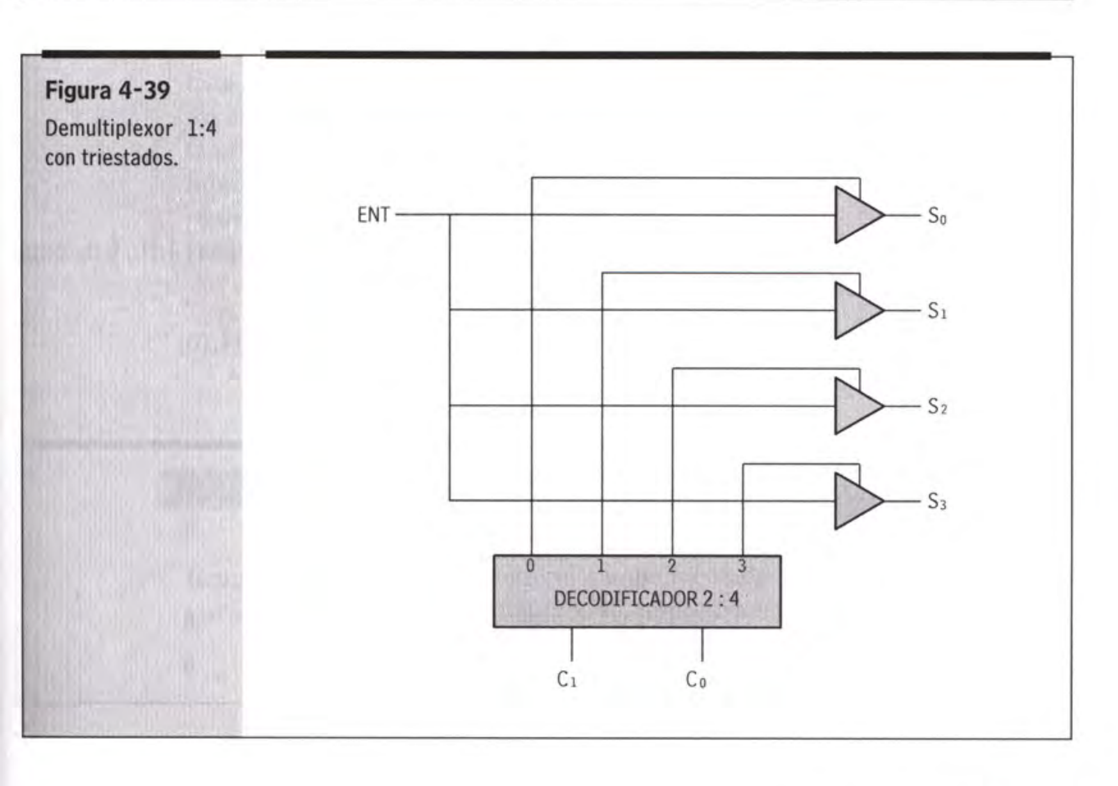


Figura 4-39
Demultiplexor 1:4 con triestados.

4.8. Comparadores

Un comparador como el de la figura 4-40 es un sistema combinacional que teniendo como entrada dos números A y B de n bits en binario puro indica si son iguales, si A es menor que B o si A es mayor que B.

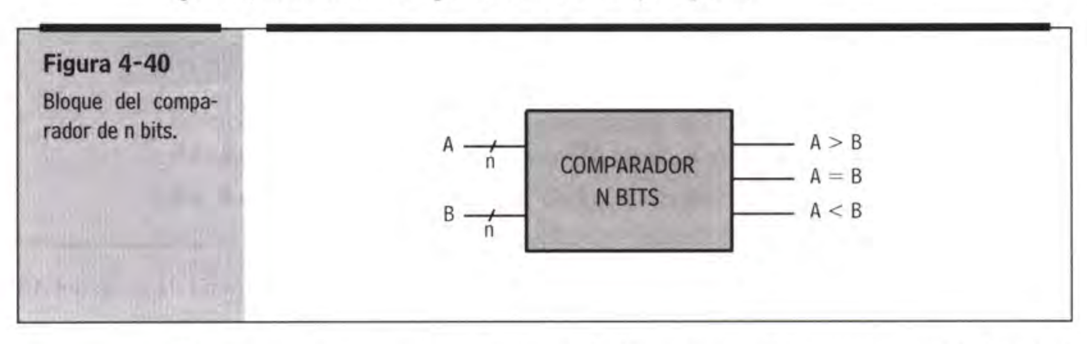


Figura 4-40
Bloque del comparador de n bits.

El comparador es un ejemplo típico de diseño extensible en su capacidad. Así, el diseño de un comparador de cuatro bits se basará en el de dos bits, y éste a su vez en el de un bit.

4.8.1. Comparador de 1 bit

Un comparador de 1 bit atiende a la tabla de verdad 4-19, donde las líneas M0, m0 y I0 significan:

- M0. Si se activa M0 significa que A es mayor que B.

  $M0 = (A > B)$

- m0. Si se activa m0 significa que A es menor que B.

  $m0 = (A < B)$

- I0. Si se activa I0 significa que A es igual a B.

  $I0 = (A = B)$

De la tabla de verdad 4-19 se obtienen las expresiones booleanas 4.10, y de éstas el circuito lógico de la figura 4-41.

$$M_0 = A_0 \cdot \overline{B_0} \qquad m_0 = \overline{A_0} \cdot B_0 \qquad (4.10)$$
$$I_0 = A_0 \cdot B_0 + \overline{A_0}\,\overline{B_0} = \overline{A_0 \oplus B_0}$$

Tabla 4-19
T-V de un comparador de 1 bit.

A0	B0	I0	M0	m0
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

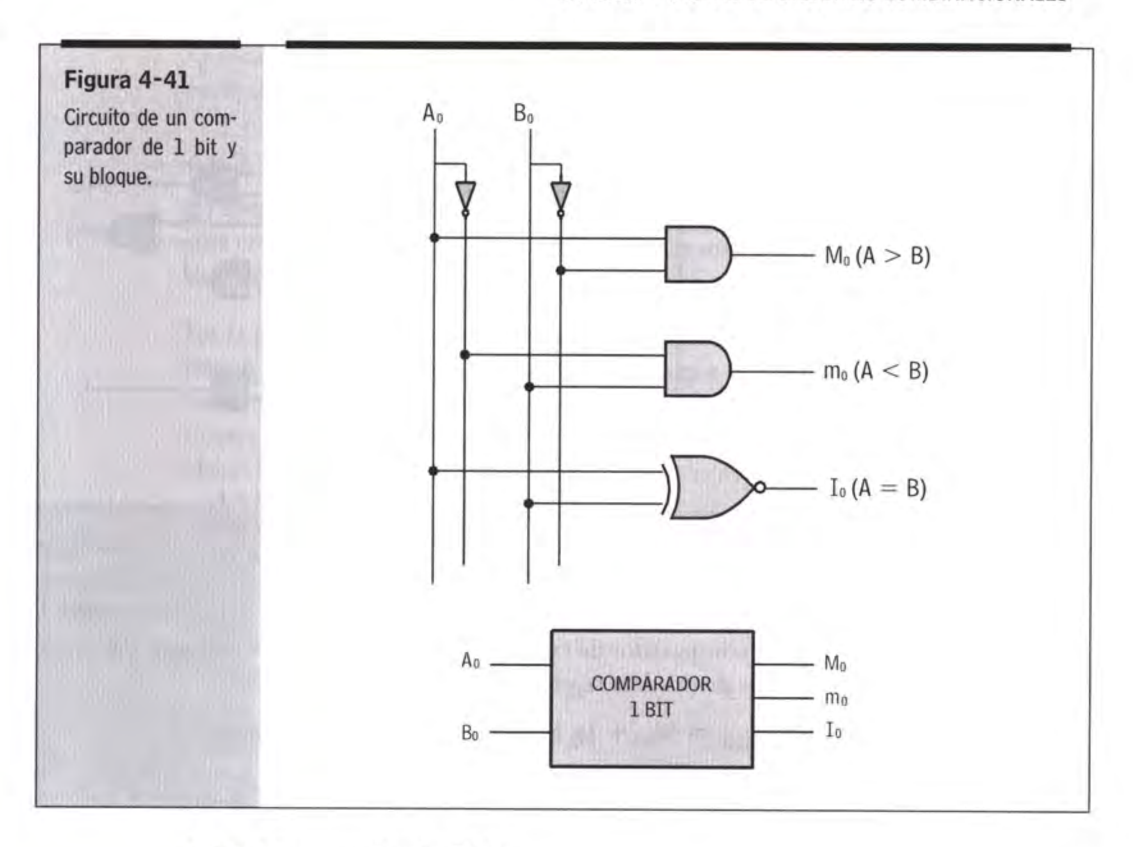


Figura 4-41
Circuito de un comparador de 1 bit y su bloque.

4.8.2. Comparador de 2 bits

Este comparador puede plantearse y resolverse como un combinacional de cuatro entradas y tres salidas: tabla de verdad, diagramas V-K, etc. Sin embargo, también podemos acercarnos a la resolución recordando que comparar dos números de n dígitos es lo mismo que comparar n veces un dígito, según el siguiente razonamiento:

- A es mayor que B, si el bit de más peso de A lo es, o si siendo éstos iguales, es mayor el de menor peso de A.

  $$M_{10} = M_1 + I_1 \cdot M_0$$

- A es menor que B si el bit de menor peso de A lo es, o si siendo éstos iguales, es menor el de menor peso de A.

  $$m_{10} = m_1 + I_1 \cdot m_0$$

- A es igual a B si son iguales los bits de más peso y los de menos peso.

  $$I_{10} = I_1 \cdot I_0$$

Reunidas las ecuaciones anteriores e implementadas en su circuito lógico resultan las ecuaciones 4.11 y la figura 4-42, respectivamente.

$$M_{10} = M_1 + I_1M_0 \qquad m_{10} = m_1 + I_1M_0$$
$$I_{10} = I_1 \cdot I_0 \qquad (4.11)$$

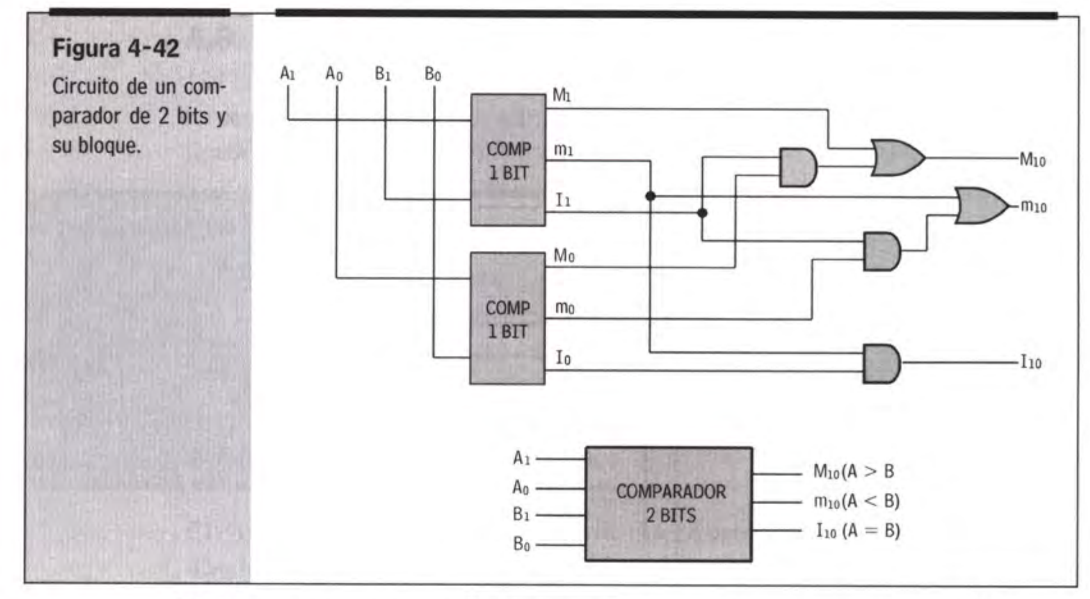


Figura 4-42
Circuito de un comparador de 2 bits y su bloque.

El diseño de un comparador de cuatro bits sigue las pautas anteriores. Partiendo de comparadores de dos bits se obtienen las ecuaciones 4.12.

$$M_{3210} = M_{32} + I_{32}M_{10} \qquad m_{3210} = m_{32} + I_{32}m_{10}$$
$$I_{3210} = I_{32} + I_{10} \tag{4.12}$$

En la implementación de la figura 4-43 podemos observar cómo la estructura del circuito es idéntica al anterior, donde simplemente se ha sustituido el comparador de un bit por el de dos bits.

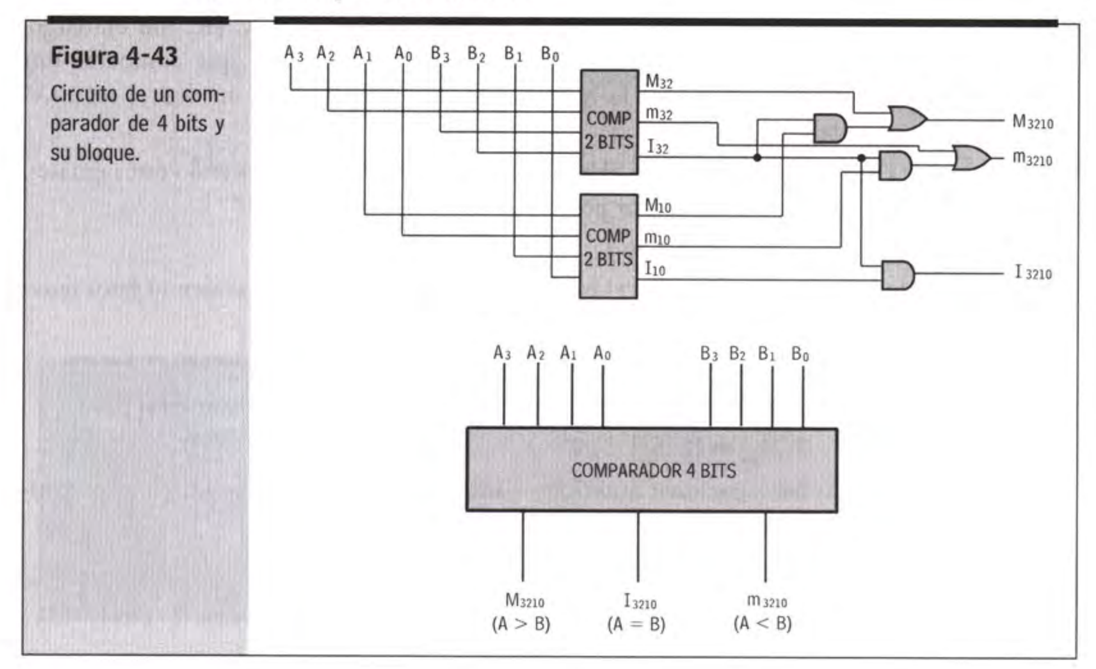


Figura 4-43
Circuito de un comparador de 4 bits y su bloque.

Queda claro que utilizando este mecanismo de replicación puede diseñarse un comparador de cualquier capacidad.

4.8.3. Comparadores MSI

El mecanismo de diseño enunciado anteriormente es válido, pero adolece de una cierta complicación, ya que para diseñar un comparador es necesario el uso de comparadores de menor tamaño y una serie de puertas.

En la práctica se utilizan circuitos comparadores de cuatro a ocho bits, que disponen de una serie de entradas auxiliares que posibilitan una fácil extensión. Las entradas auxiliares reciben información de los otros comparadores de bits de menor peso. Como muestra la tabla de verdad 4-20, dichas entradas sólo tienen efecto en la salida cuando los bits del propio comparador son iguales.

Tabla 4-20

Tabla de verdad de un comparador de 4 bits 7485.

A3,B3	A2,B2	A1,B1	A0,B0	A>BIN	A<BIN	A=BIN	A>B	A<B	A=B
A3>B3	X	X	X	X	X	X	1	0	0
A3<B3	X	X	X	X	X	X	0	1	0
A3=B3	A2>B2	X	X	X	X	X	1	0	0
A3=B3	A2<B2	X	X	X	X	X	0	1	0
A3=B3	A2=B2	A1>B1	X	X	X	X	1	0	0
A3=B3	A2=B2	A1<B1	X	X	X	X	0	1	0
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	1	0	0
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	0	1	0
A3=B3	A2=B2	A1=B1	A0=B0	1	0	0	1	0	0
A3=B3	A2=B2	A1=B1	A0=B0	0	1	0	0	1	0
A3=B3	A2=B2	A1=B1	A0=B0	X	X	1	0	0	1
A3=B3	A2=B2	A1=B1	A0=B0	1	1	0	0	0	0
A3=B3	A2=B2	A1=B1	A0=B0	0	0	0	1	1	0

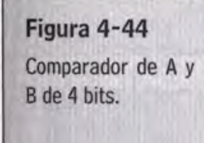
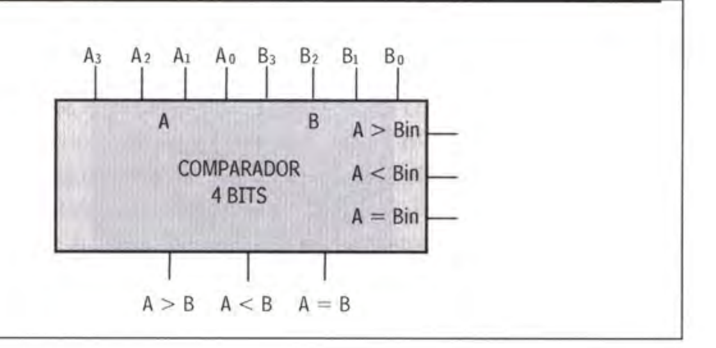


Figura 4-44

Comparador de A y B de 4 bits.



Para obtener un comparador de 4 bits atendiendo a la tabla de verdad 4-20 y al bloque 4-44, hay que forzar la entrada auxiliar A=Bin a nivel alto y las otras dos entradas auxiliares a cualquier nivel, resultando la figura 4-45.

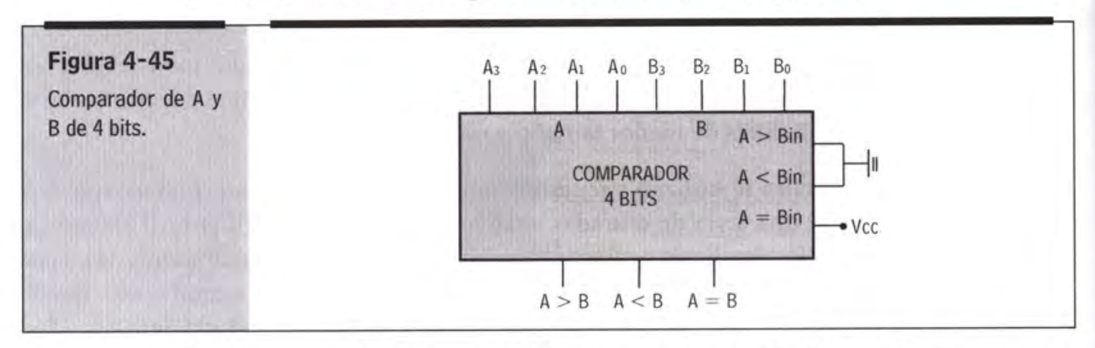


Figura 4-45
Comparador de A y B de 4 bits.

En el esquema de la figura 4-46 es un comparador de 8 bits. El planteamiento responde a que X es mayor que Y si sus cuatro bits de más peso lo son, pero si éstos son iguales A será mayor que B en función de la información recibida por los cuatro menores.

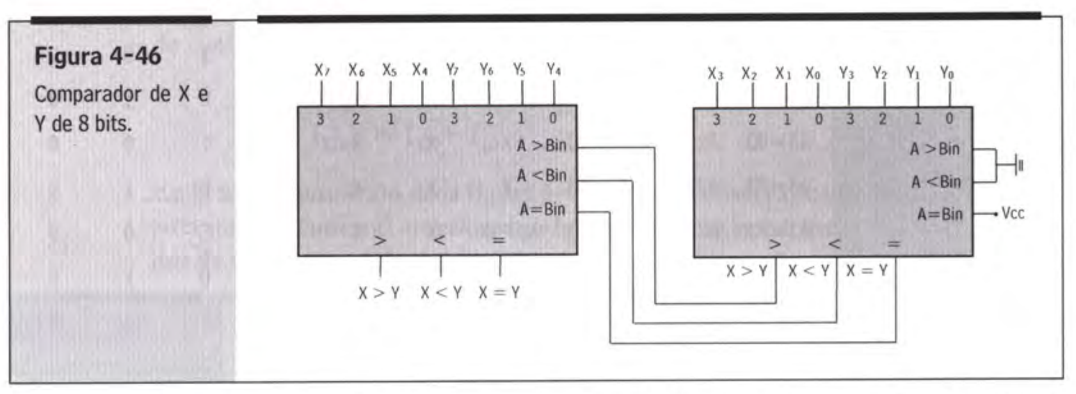


Figura 4-46
Comparador de X e Y de 8 bits.

Siguiendo parecido planteamiento se puede diseñar un comparador de siete bits sin necesidad de entradas auxiliares, siendo el bit de menor peso (o el de más peso, según la conexión) el que recibe la información de los anteriores bits, como se puede apreciar en el diseño de la figura 4-47.

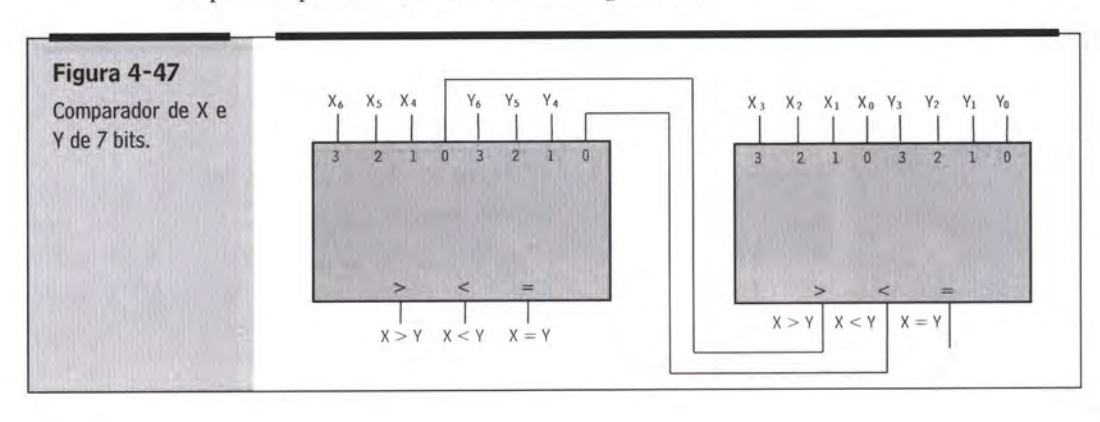


Figura 4-47
Comparador de X e Y de 7 bits.

El esquema 4-47 tiene como desventaja un menor aprovechamiento de los comparadores, ya que con dos de cuatro no se obtiene uno de ocho bits, sino de siete. En este caso un comparador de cuatro bits rinde como uno de tres, de hecho el número de bits que se puede comparar con un número de comparadores es:

$$\text{número bits} = 3 \times \text{n}^{\text{o}} \text{ comparadores} + 1.$$

Los dos esquemas 4-46 y 4-47 de ocho y siete bits presentados son en serie, así, para obtener el resultado correcto de la comparación hay que esperar a la finalización en cascada de todos los comparadores.

$$t_{comp}\text{nbits} = \text{n}^{\text{o}} \text{ comp} \times t_{comp}\text{1bit} \qquad (4.13)$$

Frente a esta evolución en serie de la comparación se puede optar por un diseño en paralelo, más complejo y costoso, pero más rápido. En este caso las entradas auxiliares pueden ser utilizadas como entradas, resultando que el comparador de cuatro bits se convierte en uno de cinco. Por ejemplo, un comparador de 24 bits en serie necesita seis comparadores y tarda 6 veces lo que uno de ellos, mientras que el esquema paralelo de la figura 4-48 utiliza también seis comparadores, pero tarda 2 x tiempo comparación.

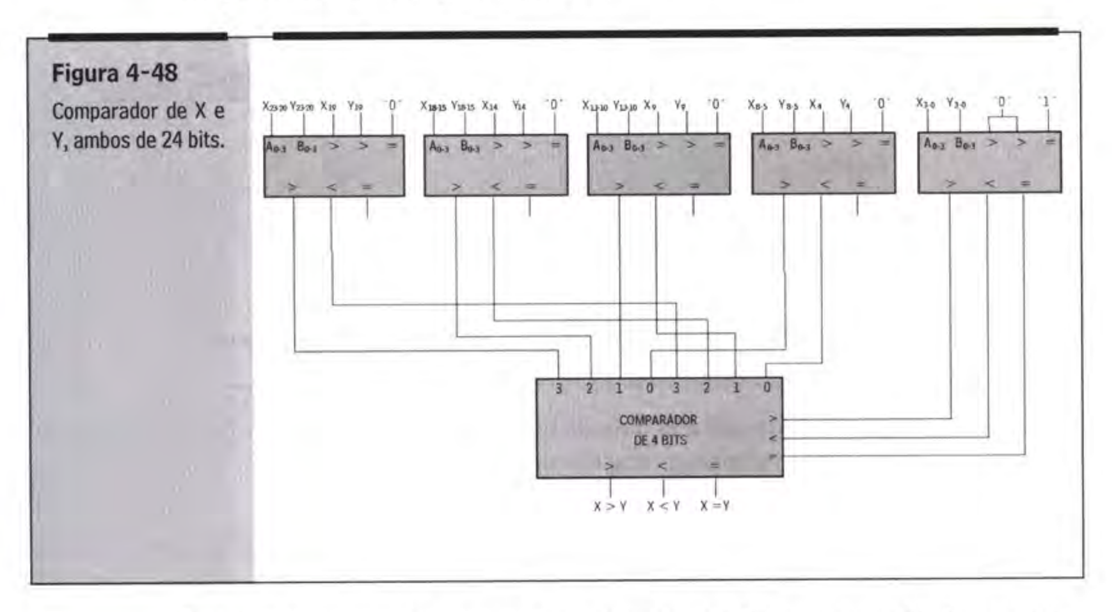


Figura 4-48

Comparador de X e Y, ambos de 24 bits.

Los circuitos presentados establecen el resultado de la comparación interpretando los números en binario puro. Si el código fuese el BCD puro no habría problema, pues el comparador de cuatro bits se convertiría en un comparador de un dígito BCD, posibilitando la comparación de varios dígitos BCD. Ahora bien, para otros códigos habría que adaptar los circuitos presentados, o incluso diseñarlos exclusivamente para ellos.

4.9. Generador/Detector de Paridad

Cuando se va a transmitir información binaria, ésta suele dividirse en bloques y cada bloque va acompañado de un bit de paridad. Este bit cumple la función de indicar si la transmisión ha sido correcta o no; cumple por tanto misión de vigilante.

Por ejemplo, la información ASCII se transmite en bloques de siete bits (un símbolo ASCII) acompañados de un bit de paridad.

El criterio de detección de errores consiste en transmitir un número de unos o ceros que siempre sea par o impar. Así, podemos distinguir distintos bits de paridad:

- Bit de paridad impar desde los unos (BPI1). El número de unos transmitidos en el bloque, incluido el bit de paridad, ha de ser impar.
- Bit de paridad par desde los unos (BPP1). El número de unos transmitidos en el bloque, incluido el bit de paridad, ha de ser par.
- Bit de paridad impar desde los ceros (BPI0). Idéntico al primero, pero respecto de los ceros.
- Bits de paridad impar desde los unos (BPP0). Idéntico al segundo, pero respecto de los unos.

Ejemplo 4-10

Determinación para distintos mensajes del bit de paridad a añadir según sea el criterio elegido.

Mensaje	Paridad impar 1	Paridad par 1	Paridad impar 0	Paridad par 0
0110110	1	0	1	0
0101010	0	1	0	1
0110	1	0	0	1
0001	0	1	1	0

La tabla anterior nos puede servir de guía a la hora de diseñar detectores/generadores.

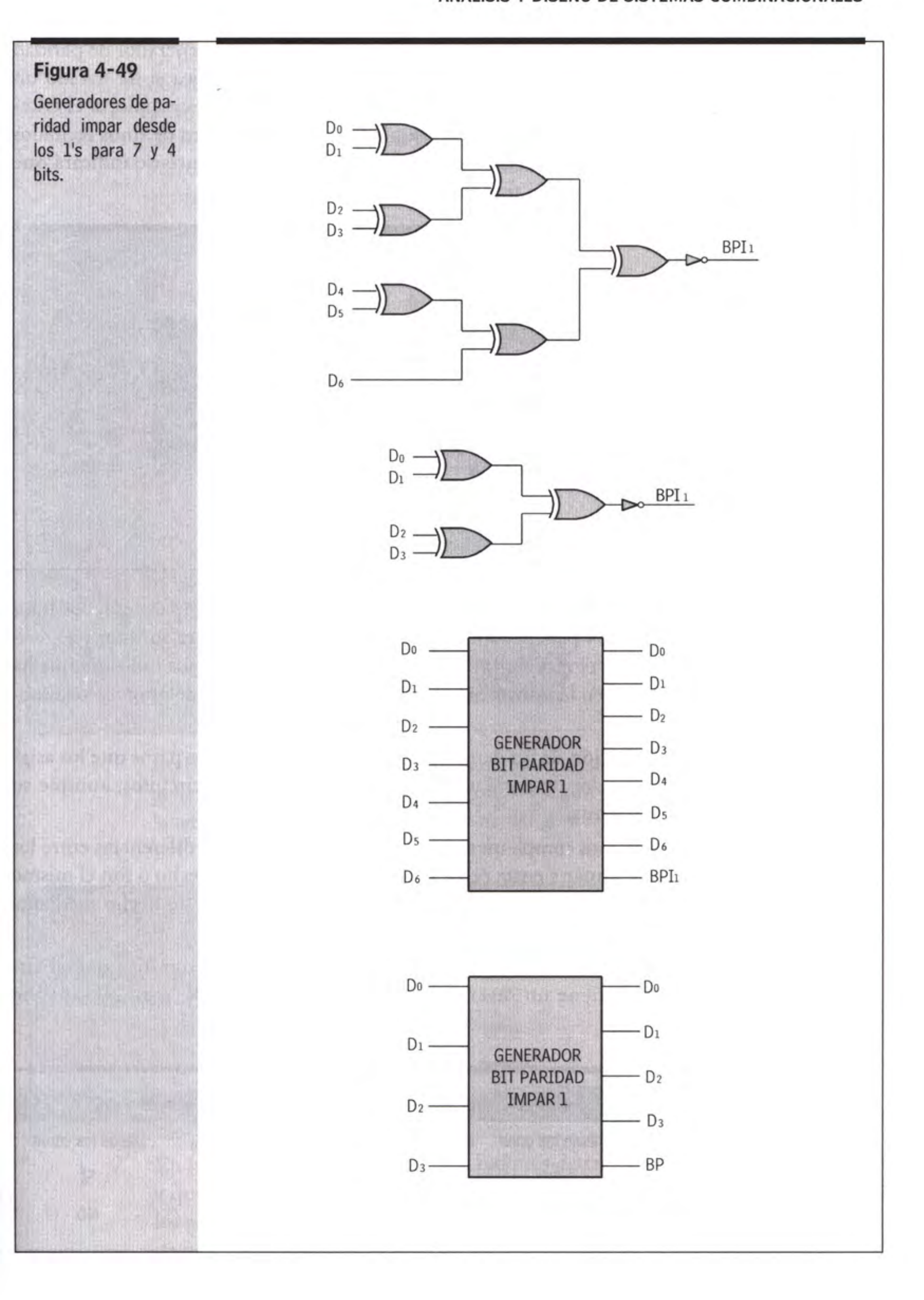
La puerta principal a la hora de implementar este circuito es la XOR. Podemos sistematizar su implementación siguiendo tres pasos:

1. Agrupamos las entradas de dos en dos en puertas XOR.
2. Las salidas de las puertas XOR se agrupan de nuevo en puertas XOR. Se repite este agrupamiento hasta conseguir una única salida.
3. Se determina, mediante un ejemplo y según el criterio de paridad elegido, la necesidad de un inversor.

Siguiendo estos pasos podemos diseñar el circuito de la figura 4-49 de un generador de paridad impar desde los unos para un mensaje de siete bits (nº impar de bits) o de cuatro bits (nº par de bits).

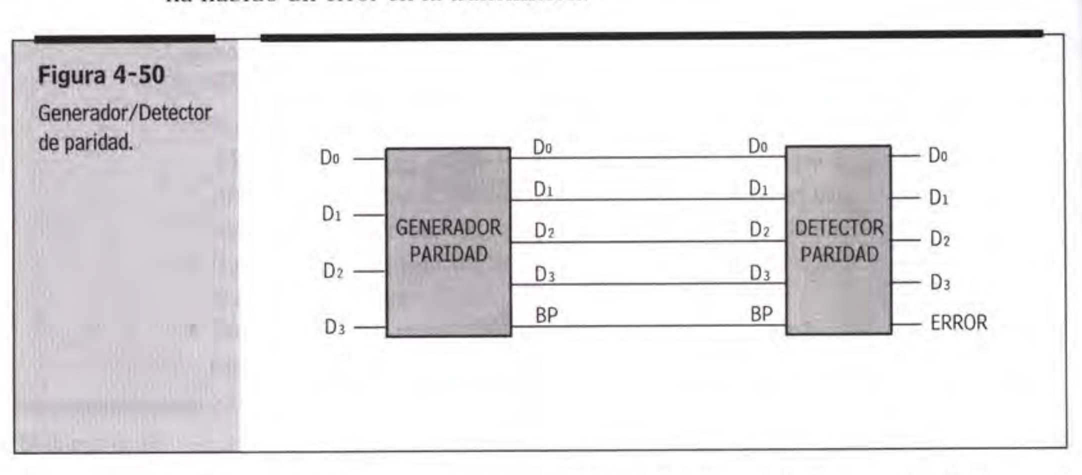
Figura 4-49

Generadores de paridad impar desde los 1's para 7 y 4 bits.



La figura 4-50 destaca que el circuito complementario del generador de paridad es el detector de paridad. Este circuito combinacional indica si ha habido un error durante la transmisión según el criterio utilizado. Por ejemplo, si el criterio de generación ha sido el impar desde los unos, el número de unos recibidos en el paquete por el detector habrá de ser impar, en caso contrario indicará que ha habido un error en la transmisión.

Figura 4-50
Generador/Detector de paridad.



Por ejemplo, si el mensaje transmitido con bit de paridad impar desde los unos fuera 0110011 1, y por la razón que fuera el cuarto bit alterara su valor y se recibiera 0111011 1; el receptor al detectar un número par de unos indicaría que ha habido un error en la transmisión, quedando bajo el control del protocolo la subsiguiente acción.

La implementación de un detector de error sigue los mismos pasos que los asignados al generador de paridad. De hecho son los *mismos* circuitos, aunque su función sea distinta.

Incidiendo en esta complementariedad cabe añadir que las diferencias entre los criterios par e impar y entre ceros y unos son mínimas, de hecho o son el mismo circuito o tienen un inversor en la salida. Remarcaremos este hecho mediante una serie de tablas.

La tabla 4-21 se refiere a generadores de paridad. Un SÍ significa que el circuito obtenido tiene un inversor en la última puerta XOR, y un NO que no lo tiene.

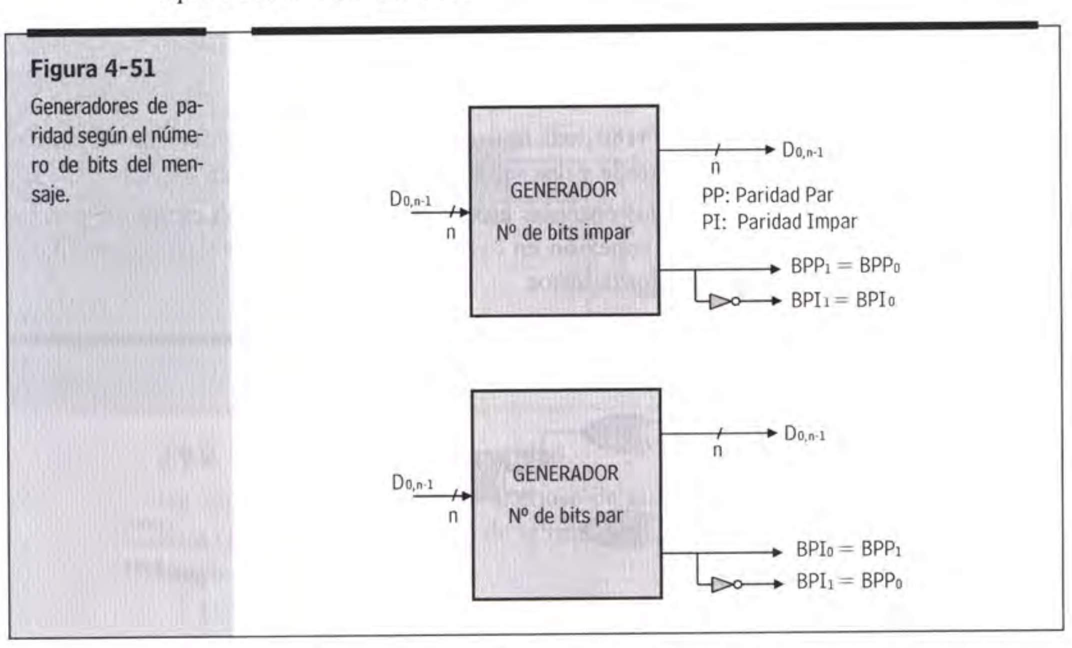
Tabla 4-21
Diseño de un generador de paridad.

	Número impar de bits		Número par de bits	
	Desde los unos	Desde los ceros	Desde los unos	Desde los ceros
Par	NO	NO	NO	SÍ
Impar	SÍ	SÍ	SÍ	NO

Es decir, si el mensaje a transmitir tuviera un número impar de bits, el generador de paridad impar sería el mismo para los ceros y para los unos; lo mismo se puede decir para el criterio de paridad par. Si el mensaje tuviera un número par de bits, el generador de bit de paridad impar desde los unos sería idéntico al correspondiente al bit de paridad par desde los ceros. La figura 4-51 resume esta equivalencia entre circuitos.

Figura 4-51

Generadores de paridad según el número de bits del mensaje.



En la tabla 4-22 relacionamos los generadores con los detectores. Se identifica qué generadores son idénticos a qué detectores según el criterio de paridad elegido. En la tabla el número de bits del mensaje del generador no incluye el bit de paridad, y el número de bits recibidos por el detector sí incluye el bit de paridad.

Tabla 4-22

Comparación entre generadores y detectores de paridad.

GENERADOR DE PARIDAD							
Nº bits del mensaje: IMPAR				Nº bits del mensaje: PAR			
BPI1	BPI0	BPP1	BPP0	BPI1	BPP0	BPI0	BPP1
BPI1	BPP0	BPP1	BPI0	BPI1	BPI0	BPP0	BPP1
Nº bits del mensaje+BP: IMPAR				Nº bits del mensaje+BP: PAR			
DETECTOR DE PARIDAD							

En la tabla 4-22 cada uno de estos bloques determina qué detectores son iguales a qué generadores. El primer bloque desde la izquierda indica que son idénticos los esquemas de un generador de paridad impar desde los unos, de un generador de paridad impar desde los ceros, y de los detectores de paridad impar desde los unos y par desde los ceros, siempre y cuando los mensajes correspondientes sean

impares (el generador no incluye el bit de paridad, el detector sí). Razonamientos iguales se pueden hacer para los subsiguientes bloques.

Estas dos tablas 4-21 y 4-22 muestran claramente cómo la distinción entre generador y detector, entre uno y cero y entre par e impar, es una simple cuestión de criterios; que por otra parte es el núcleo del circuito. Así pues, a la hora de diseñar generadores y detectores se puede partir de un circuito base, que conectado y observado de determinada manera se comportará de una u otra forma.

4.9.1. Generador/detector de paridad MSI

El circuito integrado 74180 de la figura 4-53 es un generador/detector de paridad con ocho líneas de entrada y dos salidas: paridad par e impar.

Además, dispone de dos entradas auxiliares que determinan el tipo de paridad elegido y posibilita la conexión en cascada de varios circuitos para aumentar la capacidad del generador/detector.

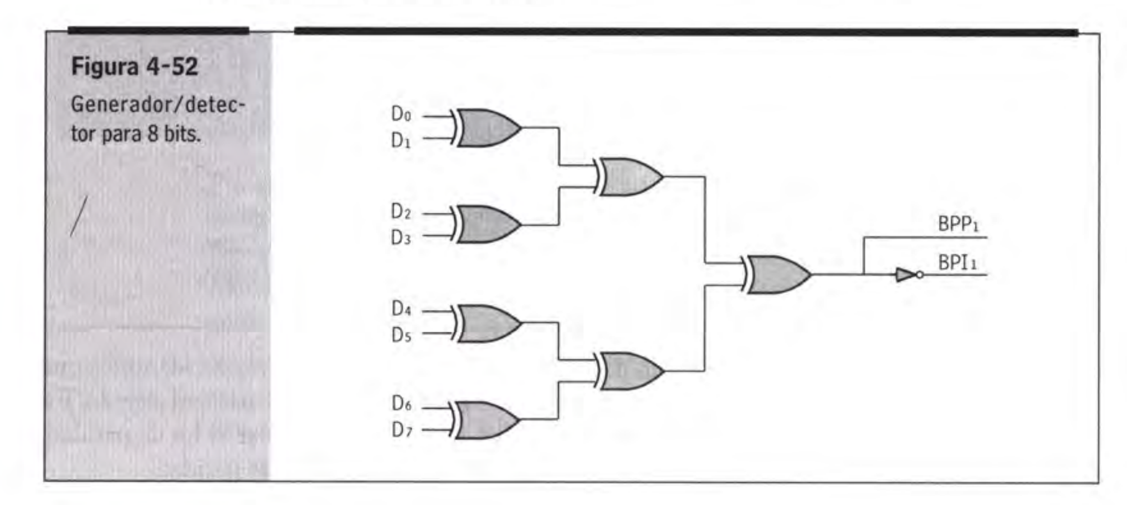


Figura 4-52
Generador/detector para 8 bits.

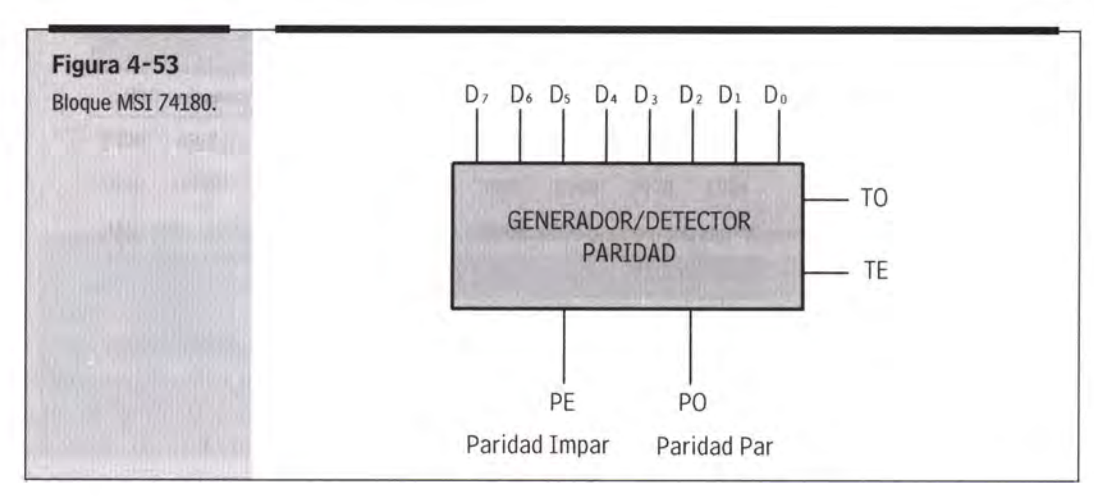


Figura 4-53
Bloque MSI 74180.

 © ITES - PARANINFO

Teniendo en cuenta el contenido de las tablas 4-21 y 4-22, y experimentando con el 74180 se puede diseñar cualquier generador/detector de paridad; ya sea con un criterio par o impar o desde los unos o desde los ceros, siendo el mensaje transmitido o el paquete recibido un número par o impar de bits. Por ejemplo, la figura 4-54 es un generador de paridad par e impar para un mensaje de 16 bits.

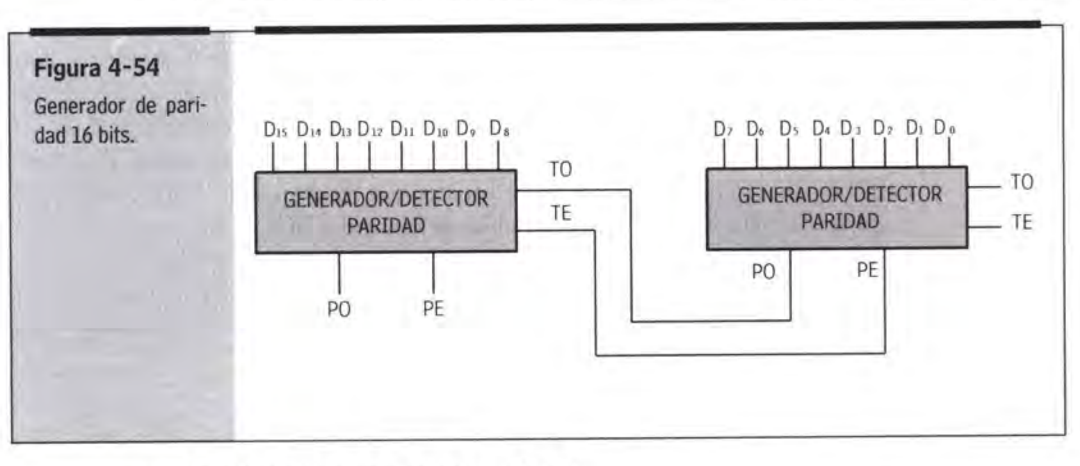


Figura 4-54
Generador de paridad 16 bits.

4.9.2. Fiabilidad del bit de paridad

Por último, cabe preguntarse si el uso de un generador/detector de paridad asegura el control de la calidad de la transmisión. La respuesta es no, por diversas causas:

- Puede que el error afecte a dos bits, con lo que el detector no determinaría error en la transmisión.

  |1101|1|Transmitido |1011|1|ERROR = 0|BPP1

- Puede que el bit modificado sea el bit de paridad, indicando el error cuando no lo haya.

  |1101|1|Transmitido |1101|0|ERROR = 1|BPP1

Resumiendo, puede que indique error cuando no lo haya, y puede que no lo indique cuando en realidad lo haya. De lo anterior se puede inferir la inutilidad de este sistema, pero no es así en la práctica. El bit de paridad se utiliza mucho en transmisiones cortas, en entornos no ruidosos, para información no importante.

4.10. Conversores de código

Un conversor es un dispositivo cuyo objetivo es transformar información numérica codificada en un determinado código a otro código. Las conversiones más típica son binario-BCD y BCD-binario, y también binario-Gray y Gray-binario.

Un conversor puede obtenerse de diversas formas:

- Diseñándolo como un circuito combinacional a nivel de bit.
- Utilizando los CI 74184 y 74185 para binario-BCD, y viceversa, respectivamente.

- Utilizando sumadores para convertir BCD a binario.
- Utilizando puertas XOR para convertir binario a Gray, y viceversa.

En el primer caso tendríamos que obtener la tabla de verdad, sus V-K, y así sucesivamente. Centrémonos en los casos más usuales.

El CI 74184 convierte seis bits BCD (4 de menos peso y dos de más peso, es decir, como máximo se puede representar el 39) en seis dígitos binarios. Por ejemplo, el 110011 en BCD (33) se convierte en 100001 en binario puro. De forma análoga, el 74185 convierte seis dígitos binarios en ocho BCD, así el 001111 en binario (15) se convierte en 00010101 en BCD. La conexión en cascada de estos CI para obtener conversores de mayor número de bits no es claramente sistematizable. En las figuras 4-55 y 4-56 se ofrecen los bloques del 74184 y del 74185 conectados como un conversor de 8 bits de binario a BCD y como un conversor de dos décadas BCD (8 bits) a binario puro.

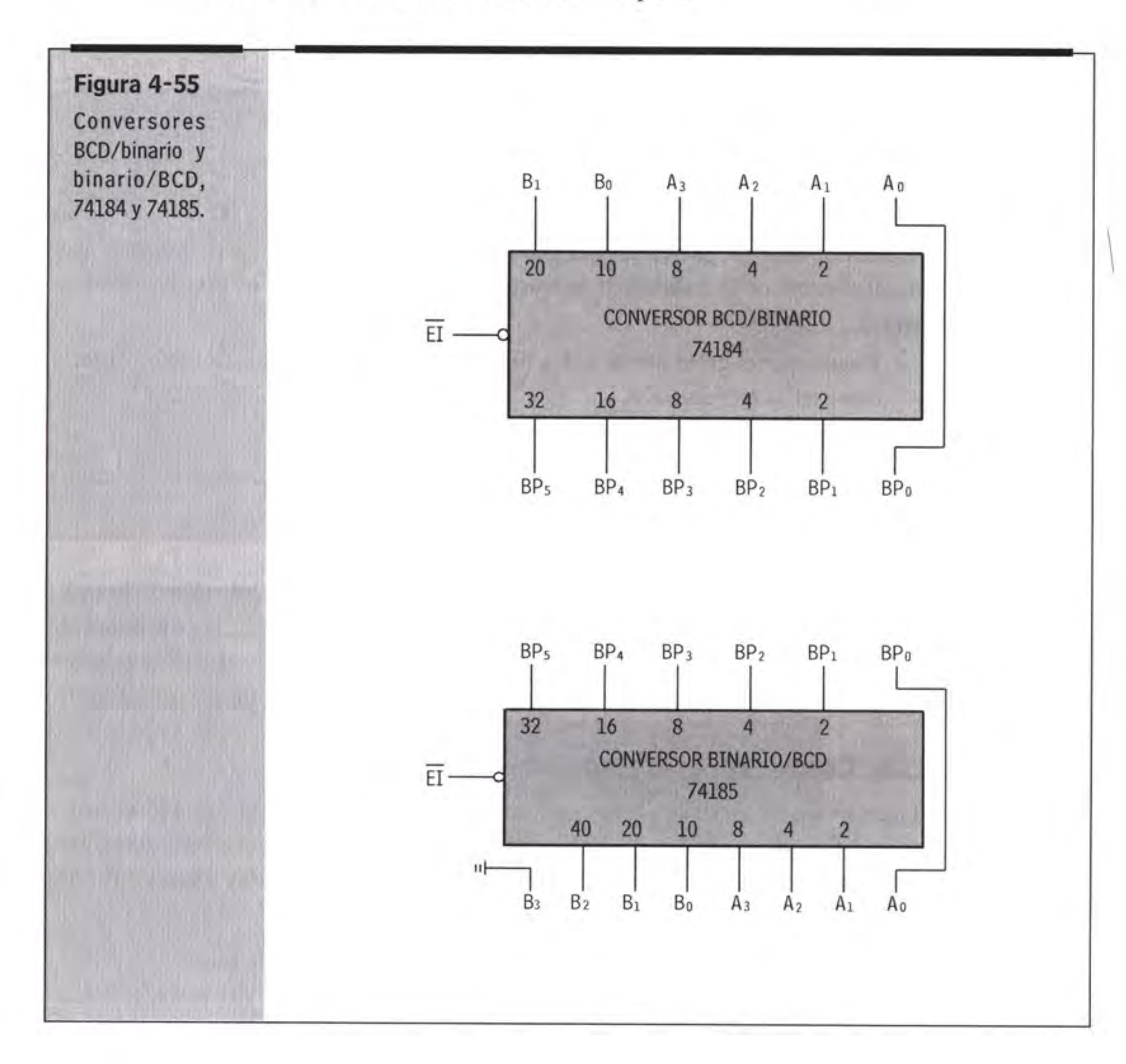


Figura 4-55
Conversores BCD/binario y binario/BCD, 74184 y 74185.

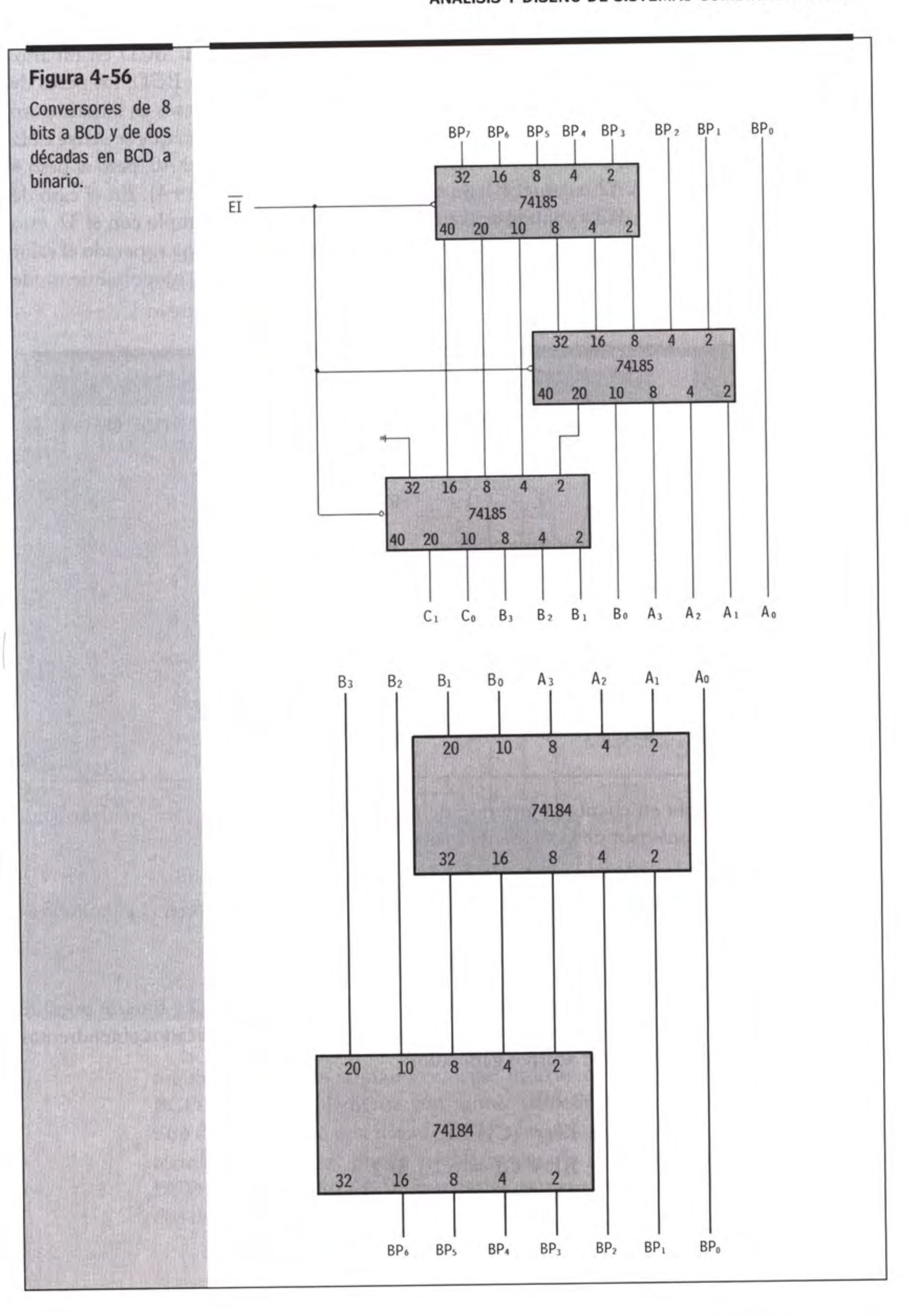


Figura 4-56

Conversores de 8 bits a BCD y de dos décadas en BCD a binario.

También está extendido el uso de sumadores para convertir BCD en binario. Para ello observemos que en un número de dos décadas en BCD, los pesos de sus ocho bits son: 1, 2, 4, 8, 10, 20, 40 y 80, mientras que los pesos en binario puro son: 1, 2, 4... y 64. La tabla 4-23 relaciona qué bits BCD hacen que se active cada bit binario. Veamos, el peso 1 en binario sólo se activa con el A0, pero el peso 4 se activa con A2 o con el B1, ya que 20 utiliza el peso 4 (16+4). En el caso de BP5, éste se activa cuando se activa el B2, o también por ejemplo con el 37, esto es, cuando la suma de los anteriores pesos (1, 2, 4, 8, 10 y 20) ha superado el valor 31; esta situación podríamos asociarla a un desborde en el comportamiento de BP4.

Tabla 4-23

Tabla para transformar dos dígitos BCD en binario puro.

	Binario Puro						
BCD	BP0 (1)	BP1 (2)	BP2 (4)	BP3 (8)	BP4 (16)	BP5 (32)	BP6 (64)
A0 (1)	1	0	0	0	0	0	0
A1 (2)	0	1	0	0	0	0	0
A2 (4)	0	0	1	0	0	0	0
A3 (8)	0	0	0	1	0	0	0
B0 (10)	0	1	0	1	0	0	0
B1 (20)	0	0	1	0	1	0	0
B2 (40)	0	0	0	1	0	1	0
B3 (80)	0	0	0	0	1	0	1

Teniendo en cuenta lo anterior, de forma intuitiva podemos sistematizar cualquier conversor de la siguiente manera:

1. Establecer la tabla que relaciona los pesos de entrada y salida.
2. Sumar aquellos bits que tienen un 1 en la tabla, teniendo en cuenta los acarreos del anterior bit.
3. Las sumas obtenidas son binario puro.

Por ejemplo, planteemos la conversión de dos décadas BCD a binario puro de siete bits. Observando la tabla 4-23 y siguiendo los pasos indicados obtendremos (entre paréntesis el acarreo generado):

$$BP0 = A0$$

$$A1 + B0 = (C1)\ BP1$$

$$A2 + B1 + C1 = (C2)\ BP2$$

$$A3 + B0 + B2 + C2 = (C3)\ BP3$$

$$B1 + B3 + C3 = (C4)\ BP4$$

B2 + C4 = (C5) BP5

B3+C5 = (C6)BP6

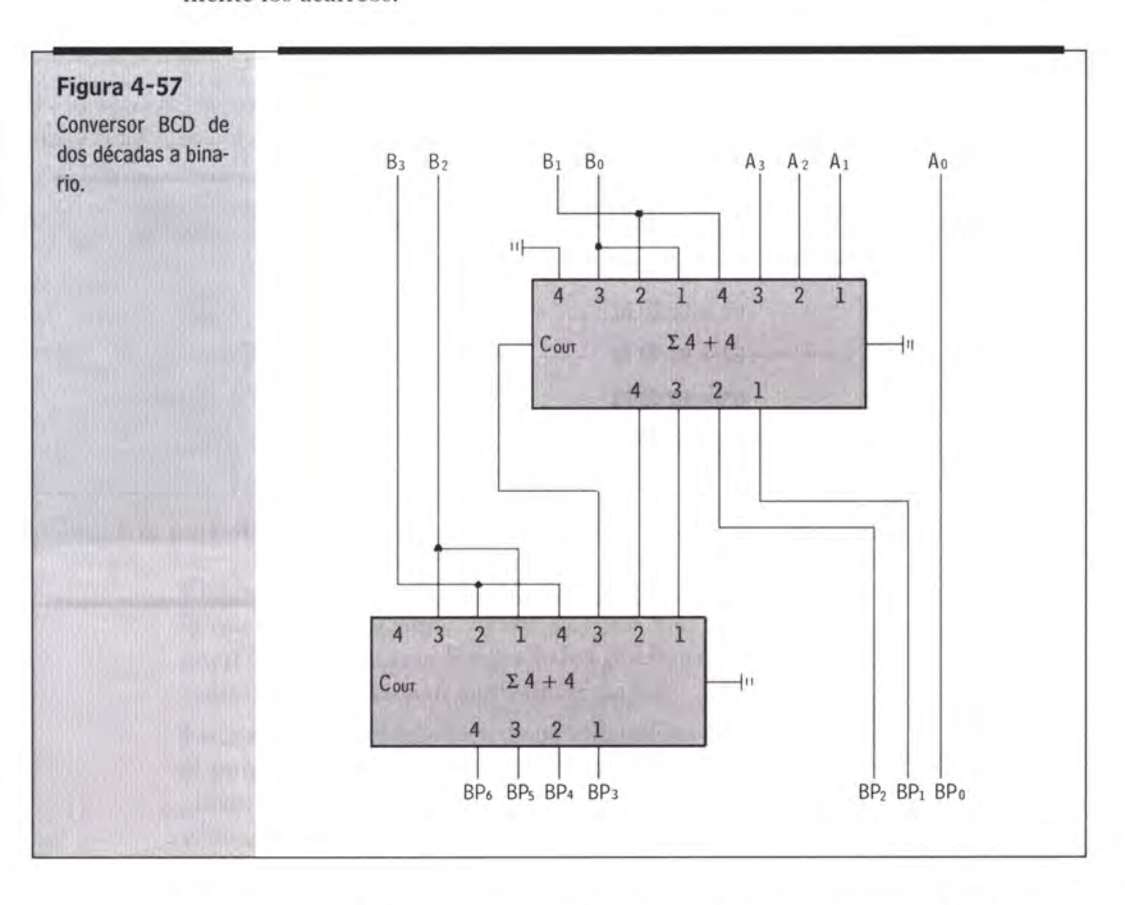
Leamos un par de expresiones de las anteriores:

- BP1 toma el valor que resulta de sumar A1+B0.
- BP2 toma el valor que resulta de sumar A2+B1 más el posible acarreo obtenido en BP1, es decir, A2+B1+C1 = BP2.

El circuito correspondiente al anterior planteamiento es el de la figura 4-57. En este caso hay que ordenar cuidadosamente las líneas para transmitir correctamente los acarreos.

Figura 4-57

Conversor BCD de dos décadas a binario.



Siguiendo con los sumadores, cabe indicar que un conversor BCD puro a BCD XS3 se implementa con tantos sumadores como décadas, donde cada uno de ellos recibe una década BCD a la que suma el valor fijo 0011. Un planteamiento similar se puede establecer para la conversión inversa, BCD XS3 a BCD puro. La figura 4-58 muestra el esquema de un conversor BCD-XS3 de dos décadas.

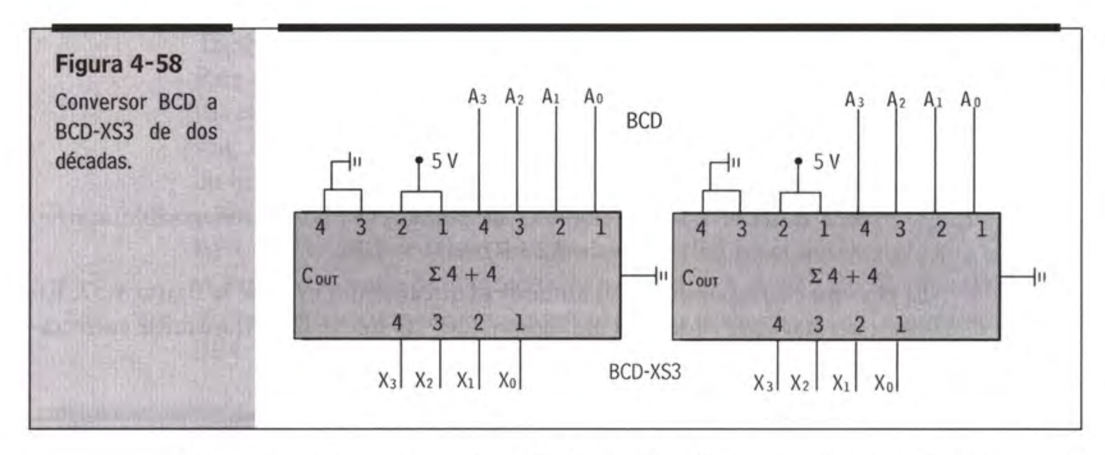


Figura 4-58
Conversor BCD a BCD-XS3 de dos décadas.

Una conversión muy común es de Gray a binario puro, y viceversa. En este caso el elemento principal es la puerta XOR y las reglas de transformación de la tabla 4-24.

Tabla 4-24
Reglas de transformación binario a Gray, y viceversa.

Binario a Gray	Gray a binario
G0 = B0	B0 = G0
G1 = B0 ⊕ B1	B1 = G0 ⊕ G1 = G1 ⊕ B0
G2 = B1 ⊕ B2	B2 = G0 ⊕ G1 ⊕ G2 = G2 ⊕ B1
G3 = B2 ⊕ B3	B3 = G0 ⊕ G1 ⊕ G2 ⊕ G3 = G3 ⊕ B2
Gn = Bn-1 ⊕ Bn	Bn = G0 ⊕ G1 .... ⊕Gn=Gn ⊕ Bn-1

Las implementaciones respectivas son muy sencillas y fácilmente extensibles, como podemos apreciar en la figura 4-59.

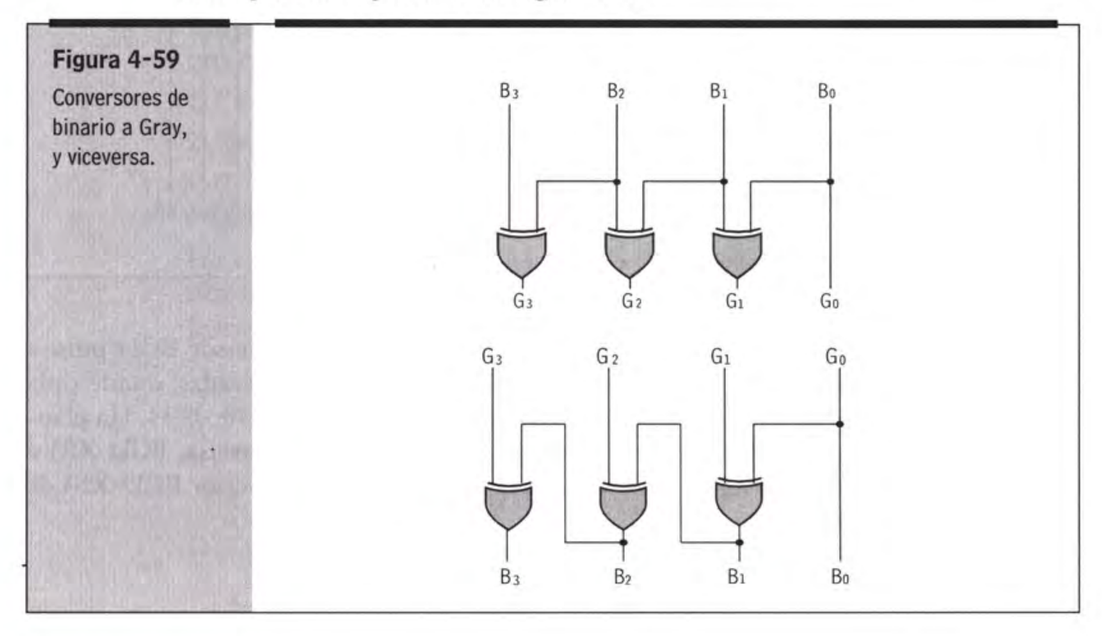


Figura 4-59
Conversores de binario a Gray, y viceversa.

4.11. Riesgos en el diseño lógico

Hasta ahora hemos considerado que para las puertas no existía retardo entre una variación en las entradas y la correspondiente en la salida. Así, si las entradas cambiaban, las salidas lo hacían instantáneamente, pero en realidad no es así; las puertas necesitan un tiempo para obtener la salida, lo que introduce un retardo entre las entradas y las salidas.

La figura 4-60 muestra la diferencia entre considerar el retardo (S') y no (S).

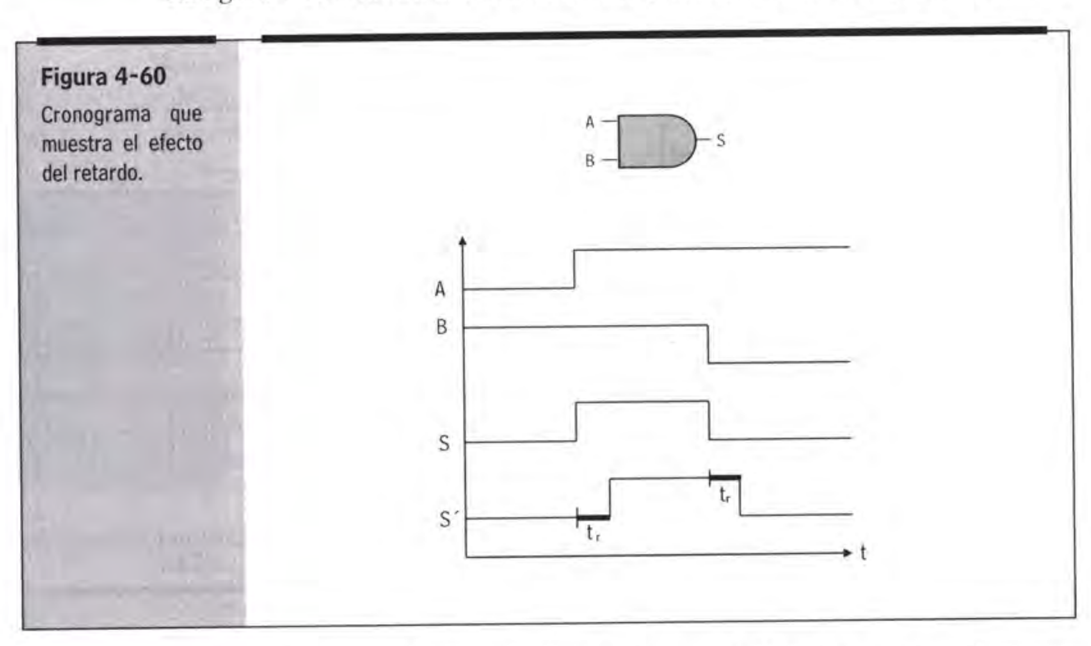


Figura 4-60
Cronograma que muestra el efecto del retardo.

El tiempo t_r de retardo depende de cada puerta y de la familia semiconductora a la que pertenezca; unas tecnologías son más rápidas que otras. Es más, entre puertas de una misma familia habrá diferencias si pertenecen a distintos fabricantes, o si evolucionan en distintas etapas.

Estos retardos son del orden de nanosegundos y podríamos considerar que desde el punto de vista práctico el efecto de estos estados transitorios (denominados glitch) puede considerarse nulo, sin embargo si el circuito tiene memoria, como veremos en capítulos posteriores, su efecto podría mantenerse perfectamente en el tiempo indefinidamente al ser memorizados, luego, tanto por interés teórico como práctico, es necesario conocer estos riesgos (denominados hazards), clasificarlos y determinar, si es posible, las formas de eliminarlos.

En el esquema y cronograma de la figura 4-61 vemos cómo la salida de f pasa por un estado transitorio erróneo (glitch) de duración t.

$$f = A \cdot B + \overline{B} \cdot \overline{C}$$

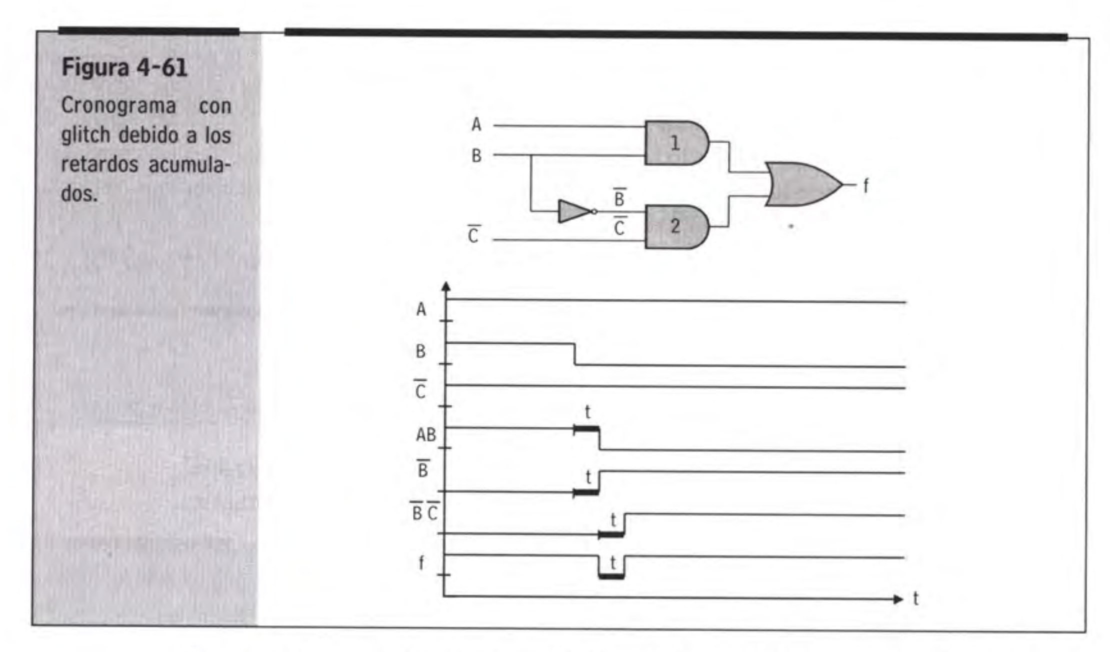


Figura 4-61

Cronograma con glitch debido a los retardos acumulados.

La situación se explica porque si la puerta 1 necesita t nanosegundos para evolucionar, la puerta 2 necesitará que previamente haya evolucionado el inversor, por tanto su evolución dura 2 x t. Esta diferencia de t nanosegundos es la responsable de la aparición del glitch.

El esquema y cronograma de la figura 4-62 presentan una situación todavía peor.

Figura 4-62

Cronograma descriptivo de un riesgo dinámico.

La primera situación (figura 4-61) es la más común, se denomina *riesgo estático*, y viene determinada por los distintos retardos asociados a cada rama del circuito implementado.

La segunda es menos usual (figura 4-62), se denomina *riesgo dinámico*. El riesgo dinámico suele estar asociado a una *mala* expresión booleana. Esta expresión, aunque represente al sistema lo hace conteniendo expresiones del tipo $A+A \cdot \overline{A}$.

Por ejemplo, en la figura 4-62, $f = A \cdot \overline{AB} + ABCD$, y para $B = C = D = 1$ resulta que $f = A \cdot \overline{A} + A$.

Resumiendo, en todas las implementaciones que, para alguna combinación de valores de entrada, puedan derivar en la forma $A + A \cdot \overline{A}$, podrá producirse un riesgo dinámico.

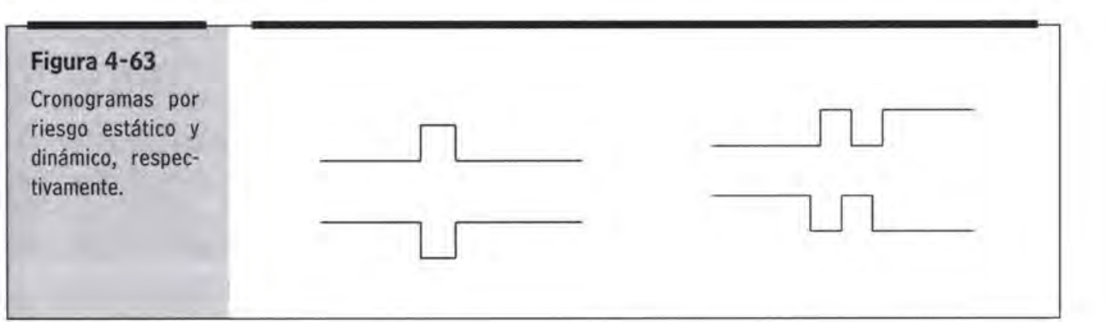


Figura 4-63

Cronogramas por riesgo estático y dinámico, respectivamente.

4.11.1. Técnicas de diseño para evitar riesgos

El origen de la aparición de los riesgos es el retardo intrínseco al uso de puertas, luego por tanto no podemos eliminarlos de raíz. Habrá que buscar mecanismos físicos o lógicos que reduzcan o eliminen su efecto.

Veamos las distintas situaciones y las posibles soluciones:

- El glitch se produce por el distinto retardo asociado a cada rama. En este caso una posible solución es añadir inversores a pares (para no variar el sentido lógico) hasta conseguir un mismo retardo en cada rama.
- Otra posible solución es no observar la salida hasta que no haya pasado un determinado tiempo desde la excursión de la entrada, y así evitar los estados transitorios. En la práctica esto supone sincronizar el circuito, pudiendo acceder a la salida sólo en aquellos instantes que aseguren su calidad. El sincronismo es muy utilizado en la práctica, pero supone un retardo general del circuito que el diseño asíncrono no tiene.
- El riesgo estático o dinámico tiene su origen en la expresión a implementar. Existen planteamientos a la hora de simplificar que aseguran la eliminación de los estados transitorios.

El sincronismo será objeto de análisis y diseño en otro capítulo. Centrémonos en la eliminación lógica de los riesgos.

Recordemos que a la hora de simplificar elegíamos el menor conjunto de lazos que asegurase cubrir la totalidad de la función a simplificar. Esto suponía el descarte de ciertos lazos que resultaban redundantes respecto del resto; es este descarte el responsable de la aparición de riesgos.

Se asocia un riesgo estático a una función si para dos casillas adyacentes del diagrama de V-K que tienen asociado un mismo valor de salida, su correspondiente lazo no pertenece a la expresión simplificada. El riesgo estático se materializa cuando la entrada pasa de una situación a la adyacente. En este caso la salida debe permanecer al mismo nivel, pero durante un tiempo pasa al estado contrario.

Si recordamos el ejemplo anterior, el glitch se ha producido cuando la entrada ha pasado de ABC = 100 a ABC = 110, ya que el lazo $A\cdot\overline{C}$ no pertenecía a la expresión simplificada. Sin embargo, si planteamos el paso ABC = 000 a ABC = 100, no se produciría riesgo estático, ya que su lazo $\overline{B}\cdot\overline{C}$ sí pertenece a f.

La solución a este tipo de riesgos estáticos pasa por incluir en la expresión booleana a implementar todos los lazos de la función, como se ve en el diagrama 4-4. Este procedimiento asegura la eliminación de riesgos, aunque no es el único método, pero renuncia a la minimización de los costes de implementación del circuito. Es decir, se plantea un compromiso entre minimalidad y asunción de riesgos.

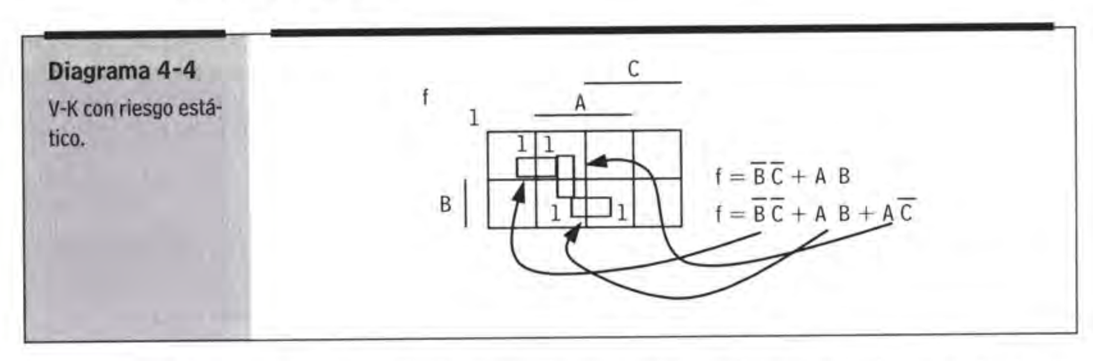


Diagrama 4-4
V-K con riesgo estático.

En oposición a la solución para riesgos estáticos, el uso de cualquier método sistemático de simplificación asegura la eliminación de las situaciones responsables de la aparición de determinados riesgos dinámicos.

4.12. Circuitos combinacionales MSI

A continuación resumimos en las tablas 4-25 y 4-26 los principales circuitos integrados de la serie 74, que implementan las puertas lógicas y los sistemas combinacionales más comunes.

© ITES - PARANINFO

Tabla 4-25

Resumen de los principales circuitos integrados con puertas.

Dispositivo	Descripción	Núm. Puertas	Núm. Entradas
7430	NAND	1	8
7420	NAND	2	4
7410	NAND	3	3
7400	NAND	4	2
7402	NOR	4	2
7427	NOR	3	3
7421	AND	2	4
7411	AND	3	3
7408	AND	4	2
7432	OR	4	2
7486	XOR	4	2
7404	NOT BUFFER	6	1
7414	SCHMITT NOT	6	1
74244	BUFFER	8	1
74245	BUFFER	8	1

Tabla 4-26

Resumen de los principales circuitos combinacionales funcionales.

Dispositivo	Funcionalidad	Tamaño	Nivel Ent.	Nivel Sal.	Líneas Auxiliares
74148	Codif. con Prioridad	8:3	Bajo	Bajo	EO(a) GS(b) [1]
74147	Codif. con Prioridad	10:4	Bajo	Bajo	
7442	Decodificador	4:10	Alto	Bajo	
7447	Decodificador	BCD 7 seg	Alto	Bajo	LT(b) RBI(b) BI/RBO(b)
74150	Multiplexor	16:1	Alto	Bajo	EN(b) [2]
74151	Multiplexor	8:1	Alto	Alto y Bajo	EN(b)
74153	Multiplexor	2 x 4:1	Alto	Alto	EN(b)
74157	Multiplexor	4 x 2:1	Alto	Alto	EN(b)
74138	Decodif./Demultiplx.	3:8 1:8	Alto	Bajo	EN1(a) EN2(b) EN3(b)
74154	Decodif./Demultiplx.	4:16 1:16	Alto	Bajo	EN1(b) EN2(b)
74155	Decodif./Demultiplx.	3:8 1:8 2 x 2:4 1:4	Alto	Bajo	EN1(b) EN2(b) EN3(b) EN4(a)
7485	Comparador	4 bits	Alto	Alto	A=B(a) A<B(a) A>B(a)
74180	Gen/Det. Paridad	8 ó 9	Alto	Bajo	PO(a) PE(a)
74280	Gen/Det. Paridad	9 bits	Alto	Bajo	
74184	BCD Binario	1 1/2 dígitos a 6 bits [3]	Alto	Alto	EI(b)
74185	Binario BCD	6 bits a 2 dígitos [4]	Alto	Alto	EI(b)

(1) (a) significa que la línea es activa por nivel alto, mientras que (b) significa por nivel bajo.
(2) Las líneas marcadas EN son líneas de Enable aunque no se denominen así en las hojas técnicas.
(3) La entrada es un número BCD de dígito y medio, o sea, hasta 39.
(4) Como máximo obtiene el 63 en la salida.

Las dos tablas anteriores no muestran la totalidad de puertas y sistemas combinacionales, sino los más representativos y orientativos para buscar el resto. En este sentido, recordar que las páginas WEB de Philips y Texas Instruments (y otras) pueden ser de gran ayuda (www.philips.com y www.ti.com).

4.13. Ejemplos de sistemas combinacionales a nivel de bit

En este apartado plantearemos y resolveremos tres sistemas combinacionales a nivel de bit.

4.13.1. Transcodificador BCD-XS3

Este sistema recibe como entrada los cuatro bits de un dígito BCD, y obtiene como salida los cuatro bits correspondientes al dígito de entrada codificado ahora en XS3.

Completemos las fases de diseño vistas al principio de este capítulo.

Variables de entrada y salida

Entrada B3-0. Cuatro bits del dígito BCD codificado en la entrada.

Salida X3-0. Cuatro bits del dígito XS3 codificado en la salida.

Tabla de verdad

Éste es el punto más delicado, aunque no por eso es difícil. Basta leer cada fila de la tabla de verdad como una pregunta, escribiendo la respuesta en la salida. Por ejemplo, la secuencia 0000 de la primera fila de la entrada es el 0 decimal según el BCD, ¿cómo se escribe el 0 decimal en XS3? Se escribe 0011. Y así para todas las filas.

Tabla 4-27

Tabla del transcodificador BCD-XS3.

	B3	B2	B1	B0	X3	X2	X1	X0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
10	1	0	1	0	X	X	X	X
11	1	0	1	1	X	X	X	X
12	1	1	0	0	X	X	X	X
13	1	1	0	1	X	X	X	X
14	1	1	1	0	X	X	X	X
15	1	1	1	1	X	X	X	X

Caso aparte son las seis últimas filas. Como las secuencias 1010 a 1111 no son BCD, podemos tomarlas como condiciones libres. En este caso la tabla de verdad es la 4-27.

Formas normales

Este paso es directo, sin más obtenemos la forma normal elegida, en nuestro caso la disyuntiva. Plantearemos la forma normal compacta, ya que la suma de minitérminos no nos interesa, pues implementaremos la expresión simplificada.

$X3 = \Sigma(5\text{-}9)$ $\quad X2 = \Sigma(1\text{-}4, 9)$

$X1 = \Sigma(0, 3, 4, 7, 8)$ $\quad X0 = \Sigma(0, 2, 4, 6, 8)$

$X = \Sigma(10\text{-}15)$

Diagramas V-K y simplificación

Los diagramas de V-K y las simplificaciones son las indicadas en los diagramas 4-5.

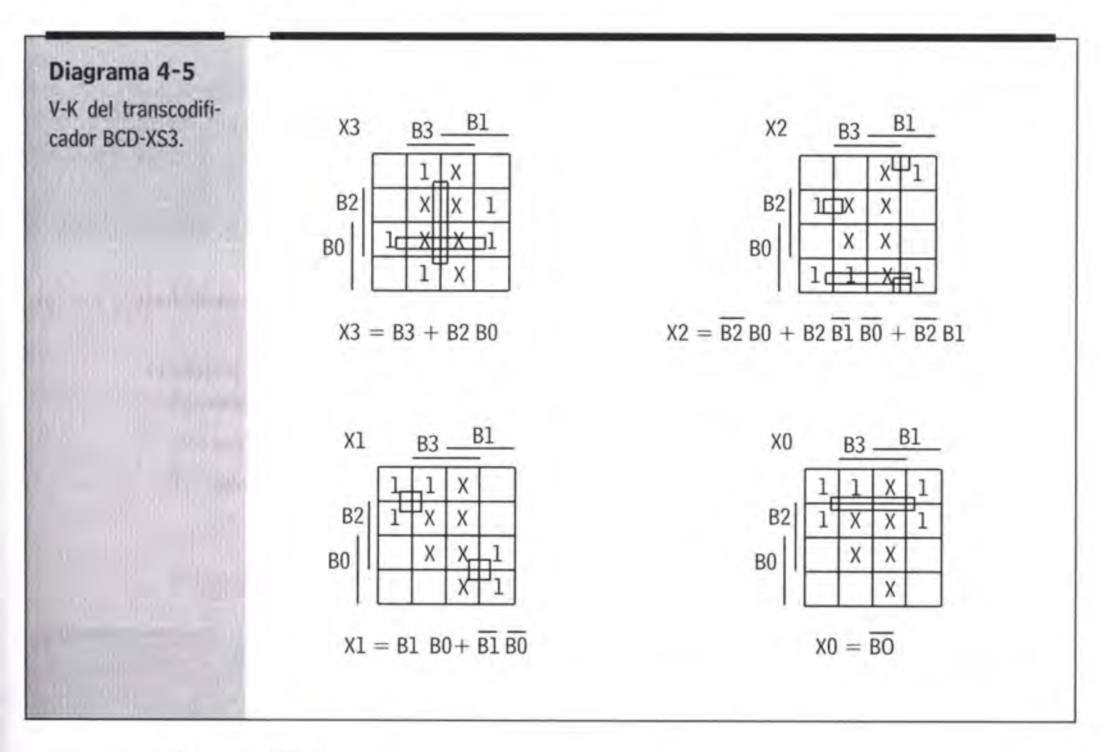


Diagrama 4-5
V-K del transcodificador BCD-XS3.

Circuito lógico

Simplemente tenemos que transformar cada expresión booleana en el circuito lógico de la figura 4-64.

Implementación mediante CI's

Para este circuito serán necesarios 4 CI's: 1 CI 7404, 2 CI 7408 y 1 CI 7432.

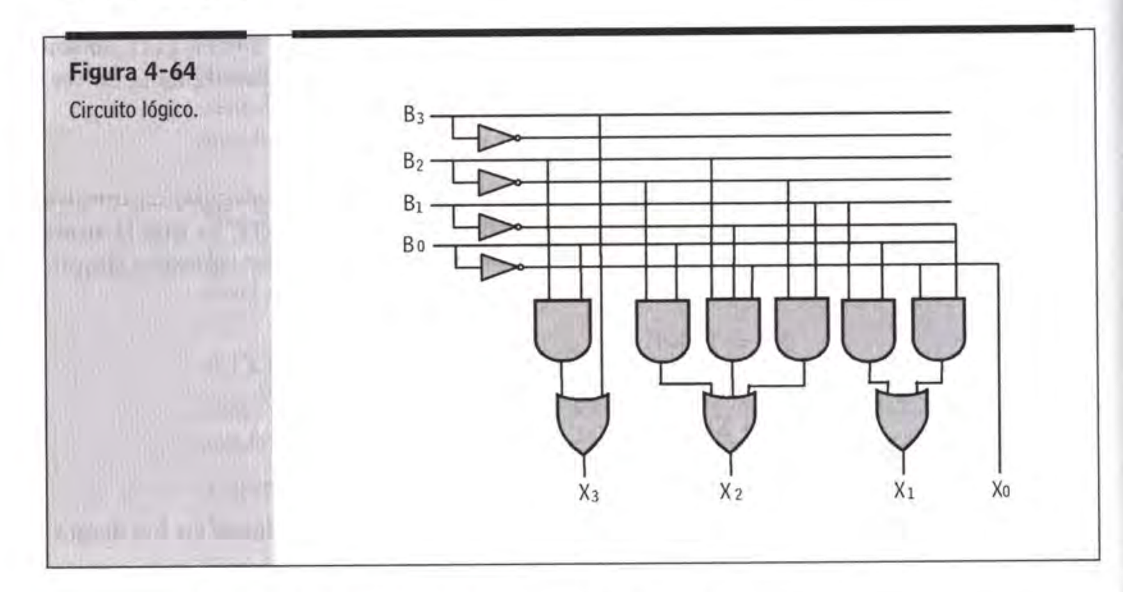


Figura 4-64
Circuito lógico.

4.13.2. Controlador de aire acondicionado

Se dispone de dos sensores de temperatura SA y SB (A se activa a los 20 °C y B a los 25) y de dos ventiladores (VA y VB) controlados por sendos interruptores (IA y IB). Se desea controlar la activación de los ventiladores en base a la siguiente regla de activación:

- Cuando se superen los 20° (A) se debe activar un ventilador, y cuando se superen los 25° se activarán ambos ventiladores.
- Un ventilador sólo se puede activar si su interruptor está activo, en caso contrario no.
- En caso de duda se activará el ventilador A, de mayor potencia.
- Los ventiladores sólo deben activarse en condiciones normales.

Variables de entrada y salida

Entrada SA. Sensor de temperatura A que se activa a los 20 °C.

Entrada SB. Sensor de temperatura B que se activa a los 25 °C.

Entrada IA. Interruptor de control del ventilador A.

Entrada IB. Interruptor de control del ventilador B.

Salida VA. Señal que activa el ventilador A.

Salida VB. Señal que activa el ventilador B.

Tabla de verdad

La tabla se completa recorriéndola fila por fila. La primera fila dice: no están activos los interruptores y no hace calor. ¿Deben o pueden activarse los ventiladores A y/o B? La respuesta es no. Y así sucesivamente hasta completar la tabla 4-28.

Tabla 4-28
Tabla del controlador de ventiladores.

	SA	SB	IA	IB	VA	VB
0	0	0	0	0	0	0
1	0	0	0	1	0	0
2	0	0	1	0	0	0
3	0	0	1	1	0	0
4	0	1	0	0	0	0
5	0	1	0	1	0	0
6	0	1	1	0	0	0
7	0	1	1	1	0	0
8	1	0	0	0	0	0
9	1	0	0	1	0	1
10	1	0	1	0	1	0
11	1	0	1	1	1	0
12	1	1	0	0	0	0
13	1	1	0	1	0	1
14	1	1	1	0	1	0
15	1	1	1	1	1	1

Las filas 4 a 7 no deberían darse (no puede haber más de 25 °C y menos de 20 °C), pero no podemos poner X, ya que dice que los ventiladores "sólo se activarán en condiciones normales". Así pues, en esos cuatro casos ambos ventiladores permanecerán inactivos.

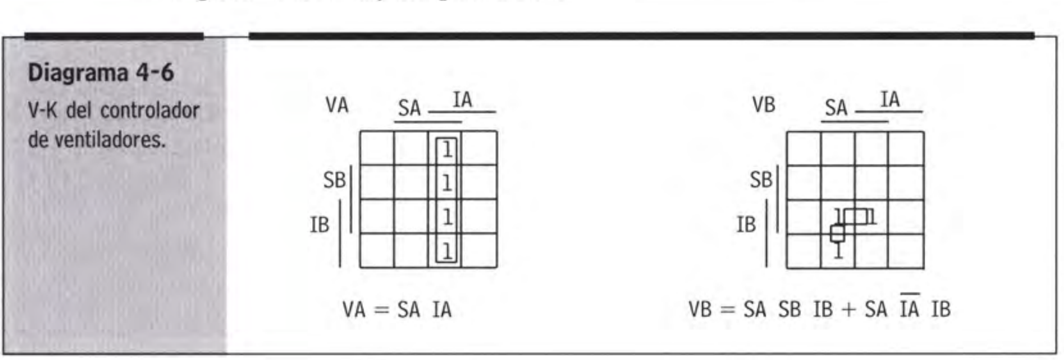
Formas normales

En este caso optaremos por la forma normal disyuntiva.

$$VA = \Sigma\ (10,11,14,15) \qquad VB = \Sigma\ (9,13,15)$$

Diagramas de V-K y simplificación

Diagrama 4-6
V-K del controlador de ventiladores.



Circuito lógico

El circuito correspondiente a las ecuaciones del diagrama 4-6 es de una gran sencillez, como muestra la figura 4-65.

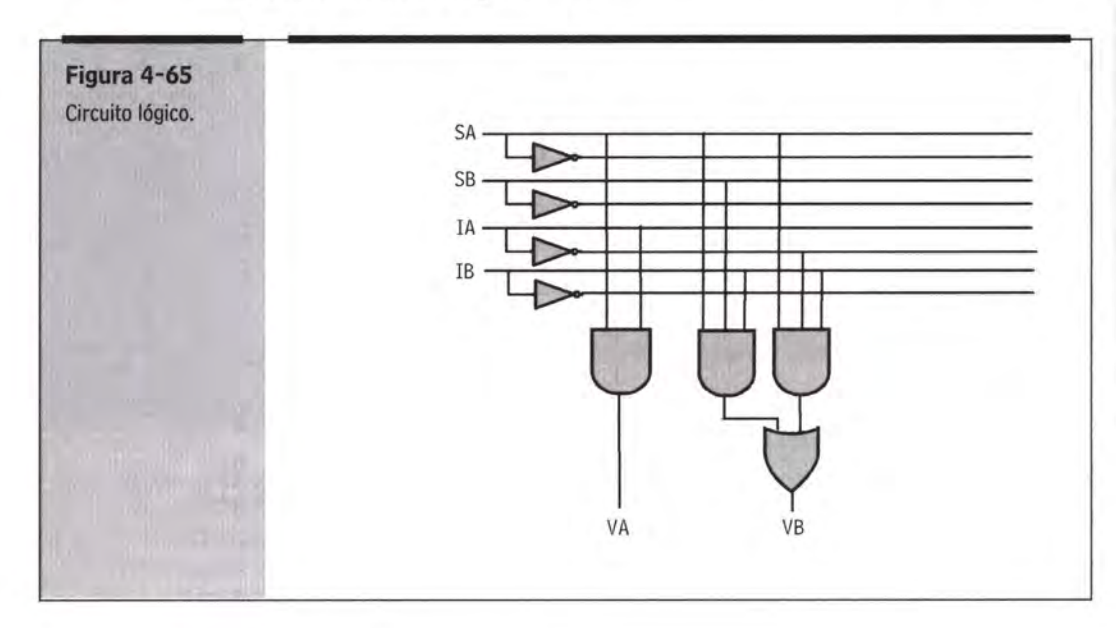


Figura 4-65
Circuito lógico.

Implementación mediante CI's

En este caso bastará con tres CI's: 1 7404, 1 7408 y 1 7432.

4.13.3. Multiplicador de dos bits

Se desea diseñar el circuito digital capaz de multiplicar dos números A y B de dos bits cada uno, codificados ambos en binario puro.

Variables de entrada y salida

Entrada A1-0. Son los dos bits de entrada del número A.

Entrada B1-0. Son los dos bits de entrada del número B.

Salida M3-0. Son los cuatro bits resultantes al multiplicar A y B.

Tabla de verdad

La primera fila (0000) dice si A es 0 y B es 0, ¿cuál es el resultado de su multiplicación? En este caso 0. Y así sucesivamente hasta completar la tabla 4-29.

Tabla 4-29
Tabla del multiplicador de 2 bits.

	A1	A0	B1	B0	M3	M2	M1	M0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0
4	0	1	0	0	0	0	0	0
5	0	1	0	1	0	0	0	1
6	0	1	1	0	0	0	1	0
7	0	1	1	1	0	0	1	1
8	1	0	0	0	0	0	0	0
9	1	0	0	1	0	0	1	0
10	1	0	1	0	0	1	0	0
11	1	0	1	1	0	1	1	0
12	1	1	0	0	0	0	0	0
13	1	1	0	1	0	0	1	1
14	1	1	1	0	0	1	1	0
15	1	1	1	1	1	0	0	1

Formas normales

En este caso optaremos por la forma normal conjuntiva.

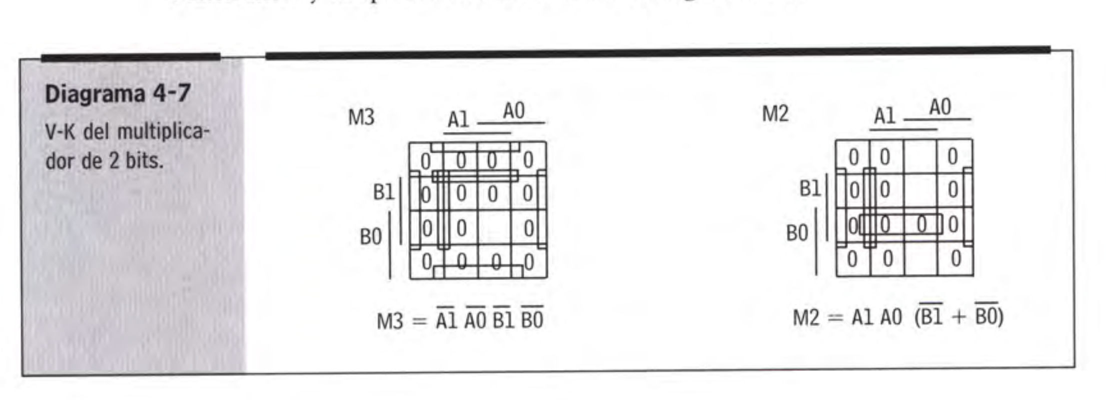
$$M3 = \Pi(0{:}14) \quad M2 = \Pi\,(0{:}9,12,13,15)$$

$$M1 = \Pi(0{:}5,8,10,12,15) \quad M0=\Pi(0{:}4,6,8{:}12,14)$$

Diagramas de V-K y simplificación

Planteamos y simplificamos los V-K en el diagrama 4-7.

Diagrama 4-7
V-K del multiplicador de 2 bits.



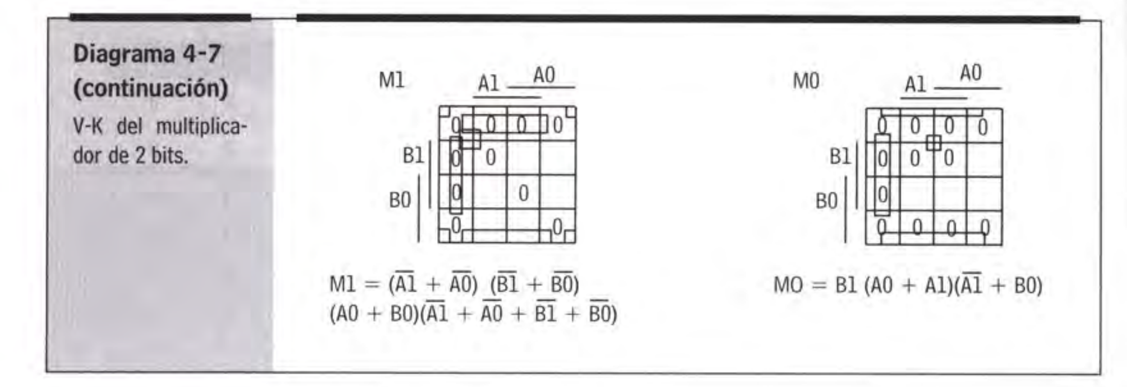


Diagrama 4-7 (continuación)
V-K del multiplicador de 2 bits.

Circuito lógico

En este caso el circuito lógico queda para el lector.

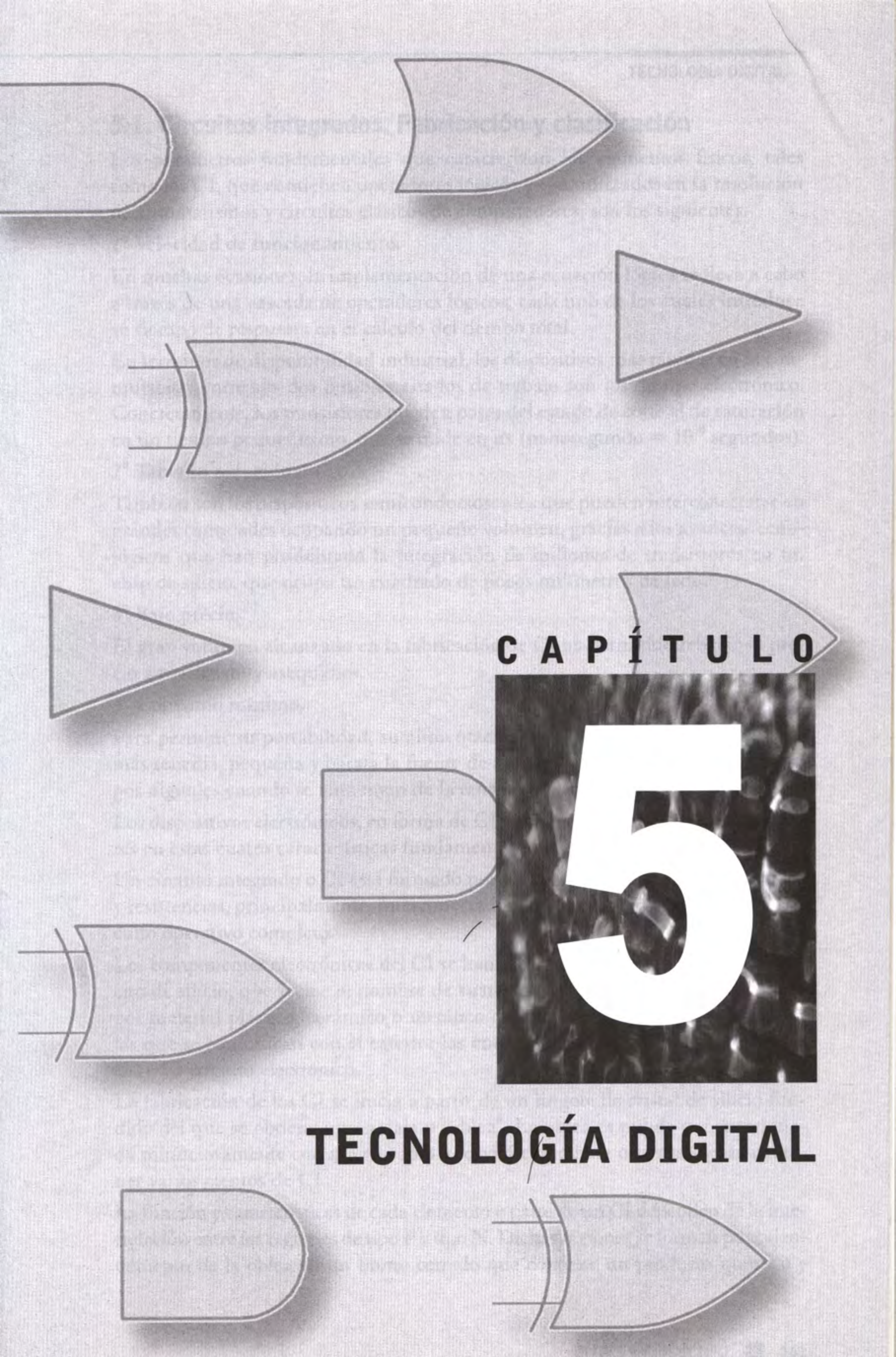
4.14. Resumen

En este capítulo hemos estudiado los sistemas combinacionales, aquellos en los que la salida sólo depende de la entrada.

A la hora de analizar o diseñar un sistema lo primero que hay que saber es si el sistema es a nivel de bit o funcional. En el primer caso los sistemas no pueden ser muy complejos, pero a cambio su diseño es sistemático.

En el caso de los funcionales los sistemas pueden ser tan complejos como se quiera, pero a cambio el diseño no es sistemático, sino basado en reglas y en el propio sistema a diseñar. Sólo diseñando se adquirirá la experiencia suficiente para afrontar con éxito estos diseños.

Recordemos también que al estudiar el capítulo hay que distinguir dos puntos de vista para cada bloque funcional. Por un lado hay que saber qué hace y para qué vale cada bloque, y por el otro hay que saber cómo lo hace. En la práctica el importante es el primero, pero éste sólo se sustentará con firmeza si contamos también con el punto de vista más teórico.



CAPÍTULO

5

TECNOLOGÍA DIGITAL

5.1. Circuitos integrados. Fabricación y clasificación

Los parámetros fundamentales que caracterizan los elementos físicos, tales como los CI, que contienen operadores lógicos y son utilizados en la resolución de automatismos y circuitos clásicos de computadores, son los siguientes:

1° Velocidad de funcionamiento.

En muchas ocasiones, la implementación de una ecuación lógica se lleva a cabo a través de una cascada de operadores lógicos, cada uno de los cuales introduce su tiempo de respuesta en el cálculo del tiempo total.

En términos de disponibilidad industrial, los dispositivos más rápidos en la conmutación entre sus dos posibles estados de trabajo son los de tipo electrónico. Concretamente, los transistores pueden pasar del estado de corte al de saturación en un tiempo pequeñísimo, que se mide en **ns** (nanosegundo = 10^{-9} segundos).

2° Tamaño reducido.

También son los dispositivos semiconductores los que pueden interconectarse en grandes cantidades ocupando un pequeño volumen, gracias a los avances tecnológicos que han posibilitado la integración de millones de transistores en un **chip** de silicio, que ocupa un cuadrado de pocos milímetros de lado.

3° Bajo precio.

El gran volumen alcanzado en la fabricación de CI ha permitido rebajar su precio a niveles muy asequibles.

4° Consumo mínimo.

Para permitir su portabilidad, su alimentación con baterías autónomas y hacer más sencilla, pequeña y barata la fuente de alimentación que precisan los equipos digitales cuando se alimentan de la tensión de la red.

Los dispositivos electrónicos, en forma de CI, reúnen el mejor promedio de valores en estas cuatro características fundamentales.

Un circuito integrado o CI está formado por un conjunto de transistores, diodos y resistencias, principalmente, interconectados entre sí para implementar un circuito operativo completo.

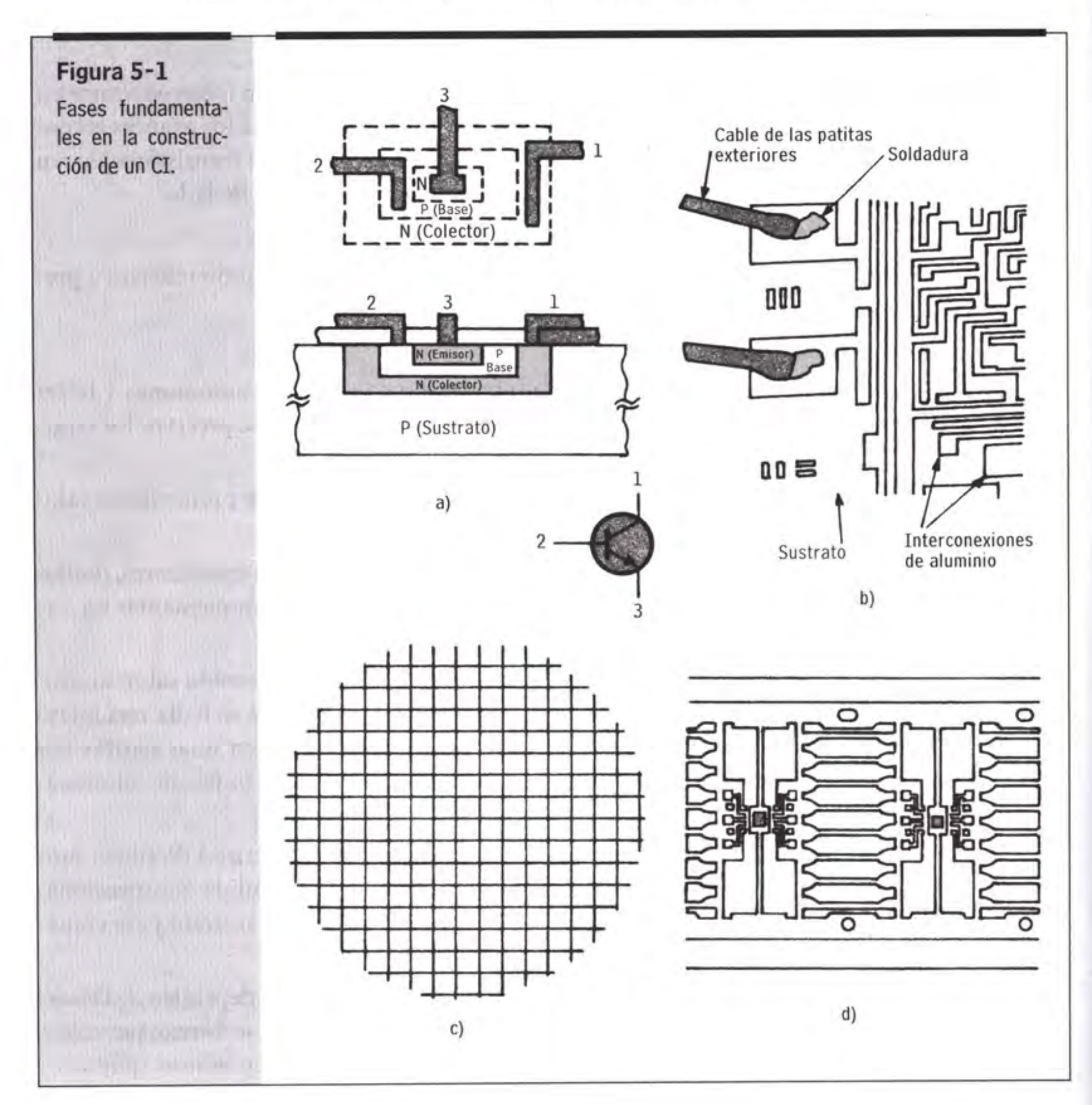
Los componentes electrónicos del CI se han fabricado y conectado sobre un trocito de silicio, que recibe el nombre de **sustrato** o **chip**. Éste se halla recubierto por material plástico, cerámico o metálico del que sobresalen unas **patitas** por las que se comunican con el exterior las entradas, salidas y tomas de alimentación del circuito electrónico.

La fabricación de los CI se inicia a partir de un lingote de cristal de silicio fundido del que se obtiene una rodaja u "oblea". La oblea es pulida e inspeccionada minuciosamente con microscopio y servirá como base o sustrato para contener varios cientos de CI.

La función y características de cada elemento o parte de un CI dependen de la interrelación entre las regiones de tipo P y tipo N. Dichas regiones se forman por calentamiento de la oblea en un horno cerrado que contiene un producto químico y

gaseoso trivalente o pentavalente. La elevada temperatura hace que los átomos del gas se difundan y penetren en la oblea cambiando sus características eléctricas. Ordenando adecuadamente la formación de las capas de material tipo P y tipo N, se crean transistores, diodos, resistencias, etc. Por ejemplo, un transistor NPN como el mostrado en la figura 5-1 a), está constituido por la difusión de una capa N, seguida de una capa P y, finalmente, una capa N, una encima de la otra. Los componentes del chip están unidos entre sí por tiras de aluminio, como se muestra en la figura 5-1 b), que se han depositado en la superficie de la oblea, metalizada previamente. Terminada la fabricación, la oblea se corta en los diferentes CI simples, según se muestra en la figura 5-1 c). Finalmente, el chip se monta en una cápsula apropiada y se sueldan las zonas metalizadas a las patitas que salen de la cápsula y comunican las entradas y salidas del circuito electrónico interno.

Figura 5-1
Fases fundamentales en la construcción de un CI.



5.1.1. Transistores bipolares y unipolares

Los CI se clasifican en bipolares y unipolares de acuerdo con el tipo de transistor que se utilice. Los CI bipolares están construidos con transistores que necesitan dos polarizaciones para su funcionamiento y pueden ser del tipo NPN y PNP. Los CI unipolares utilizan transistores de efecto de campo, tipo MOS, cuyo funcionamiento sólo exige una tensión de polarización para sus electrodos.

Los transistores bipolares, NPN o PNP, están constituidos por dos capas de material semiconductor de un tipo, separadas físicamente por una capa de material semiconductor de tipo opuesto. Por ejemplo, un transistor NPN se construye a base de dos capas tipo N, separadas por una del tipo P. En este tipo de transistores, **la magnitud de la corriente electrónica** que circula desde el emisor hasta el colector está controlada por la polarización de la capa central o base. Es decir, hay dos polarizaciones: una de ellas se aplica entre el emisor y el colector (es la más importante y la que provoca la corriente que se utilizará como salida) y otra polarización secundaria (y de muy bajo valor), que se aplica entre el emisor y la base, y gobierna el paso de la corriente entre el emisor y el colector. De esta forma, la corriente del colector IC, que es la de salida, es función de la corriente de la base IB, que es la de entrada. Figura 5-2.

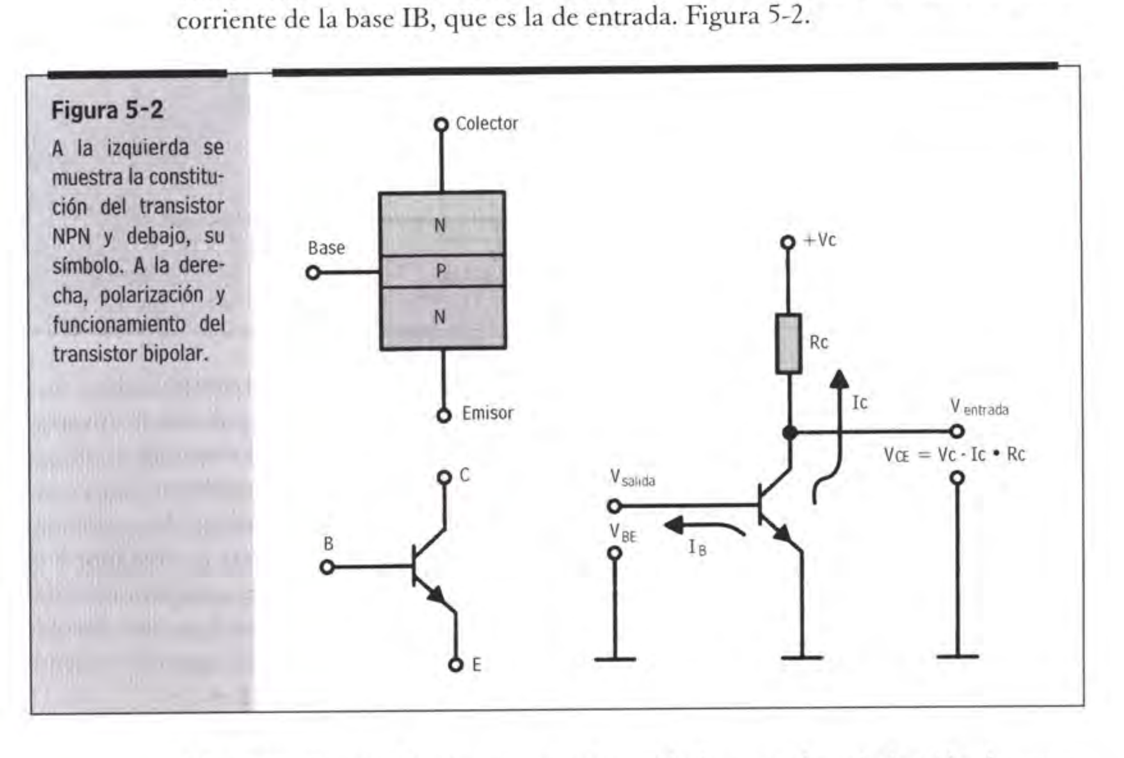


Figura 5-2

A la izquierda se muestra la constitución del transistor NPN y debajo, su símbolo. A la derecha, polarización y funcionamiento del transistor bipolar.

En los transistores bipolares, pequeñas variaciones en la polarización de entrada (base-emisor), provocan importantes variaciones en la salida (emisor-colector), consiguiendo un efecto amplificador, que no se usa en Electrónica Digital, al hacer trabajar al transistor en conmutación: **CORTE** ($I_C = 0$) y **SATURACIÓN** (I_C máxima).

El transistor unipolar, MOS (Metal-Óxido-Semiconductor), consta de dos zonas del mismo tipo de semiconductor (N o P), interconectadas por un estrecho canal de semiconductor de tipo opuesto. El paso de corriente entre las dos primeras zonas lo controla el canal según la tensión de polarización que se aplica a una zona metalizada existente sobre él, pero aislada del mismo por una capa de óxido de silicio. En la figura 5-3 se muestra la estructura interna de un transistor MOS, de canal N. Sobre el sustrato de silicio de tipo N, se han depositado dos difusiones de tipo P, que conforman sus dos electrodos principales entre los que circula la corriente de salida, llamados "drenador" y "surtidor". Las dos zonas P quedan unidas por la capa aislante de óxido de silicio, sobre la que se ha metalizado una capa de aluminio, que conforma el tercer electrodo del transistor, el "graduador".

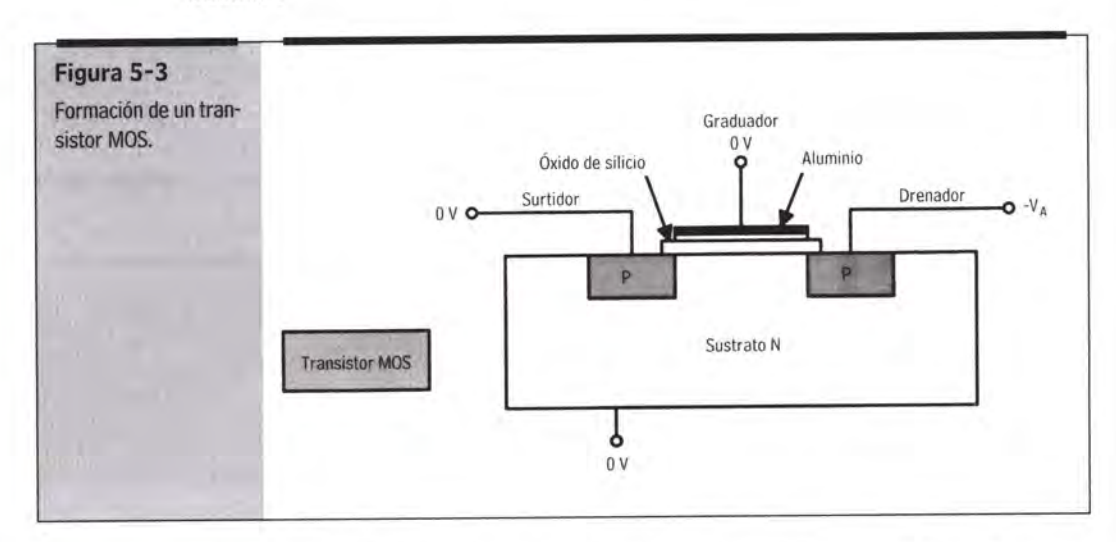


Figura 5-3
Formación de un transistor MOS.

Si el sustrato, el graduador y el surtidor se conectan a masa y el drenador a una tensión negativa $-V_A$, el diodo drenador-surtidor queda polarizado inversamente por lo que sólo podrá circular una débil corriente del surtidor al drenador. Sólo existe en este transistor una polarización principal. Si, en estas condiciones, se va aumentando la tensión negativa del graduador, éste repelerá, cada vez más, a los electrones del sustrato N existente entre el drenador y el surtidor, hasta provocar una notable ausencia de portadores mayoritarios del sustrato, convirtiéndose en tipo P dicha zona de influencia. Con ello desaparece la unión N-P polarizada inversamente y se permite el paso de corriente entre surtidor y drenador tal como se muestra en la figura 5-4.

La curva característica de un transistor MOS relaciona la corriente que circula entre surtidor y drenador con la tensión que se aplica al graduador. La anchura o tamaño del canal que se forma entre drenador y surtidor (y que determina la corriente de paso) depende de la tensión del graduador. No circula corriente hasta que se alcanza una tensión mínima en el graduador, llamada "tensión de umbral", $\mathbf{V_{TH}}$, como se representa gráficamente en la figura 5-5.

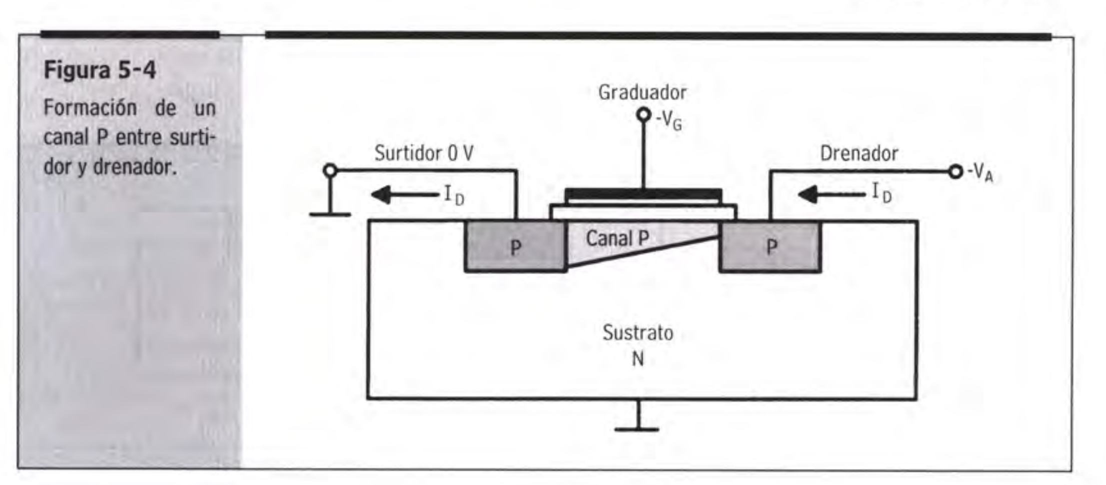


Figura 5-4
Formación de un canal P entre surtidor y drenador.

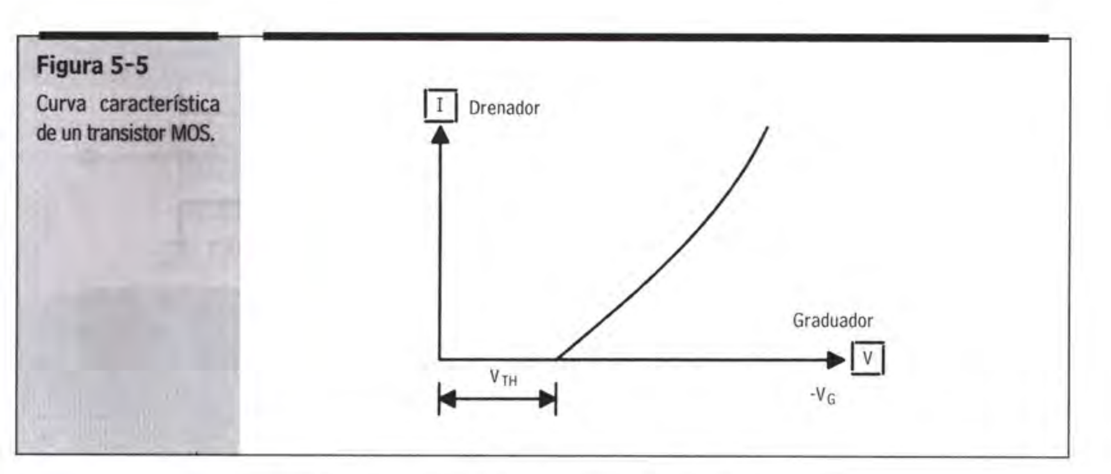


Figura 5-5
Curva característica de un transistor MOS.

Como el graduador está aislado por el óxido de silicio, apenas consume corriente, con lo que el control de la corriente principal (surtidor-drenador) se realiza sin consumo de corriente de entrada, a diferencia de lo que sucedía con los transistores bipolares.

Si en un sustrato tipo P se realizan dos difusiones tipo N, para drenador y surtidor, y sobre el canal entre ambas se deposita óxido de silicio y una capa de aluminio para el graduador, se obtiene un "transistor MOS de enriquecimiento de canal N", o abreviadamente, NMOS. Tiene el mismo comportamiento que el transistor PMOS pero diferentes polarizaciones. Figura 5-6.

En la figura 5-7 se muestran los símbolos empleados para representar los transistores de enriquecimiento.

Hay otro tipo de transistores unipolares, que se llaman de "empobrecimiento". Son similares a los de enriquecimiento, pero en su fabricación se crea un canal entre el surtidor y el drenador, cuya actuación se manifiesta incluso cuando el graduador carece de tensión. En realidad, la misión del graduador es disminuir la anchura del canal.

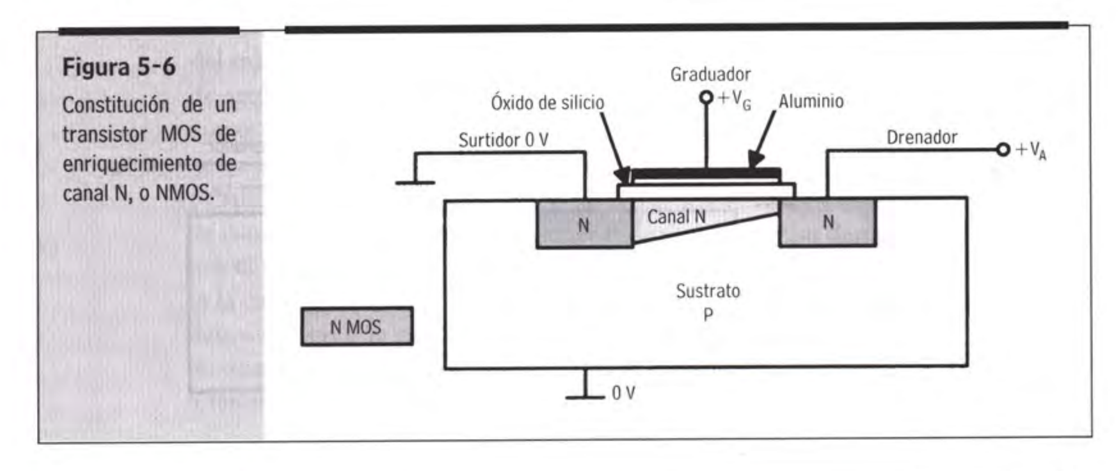


Figura 5-6
Constitución de un transistor MOS de enriquecimiento de canal N, o NMOS.

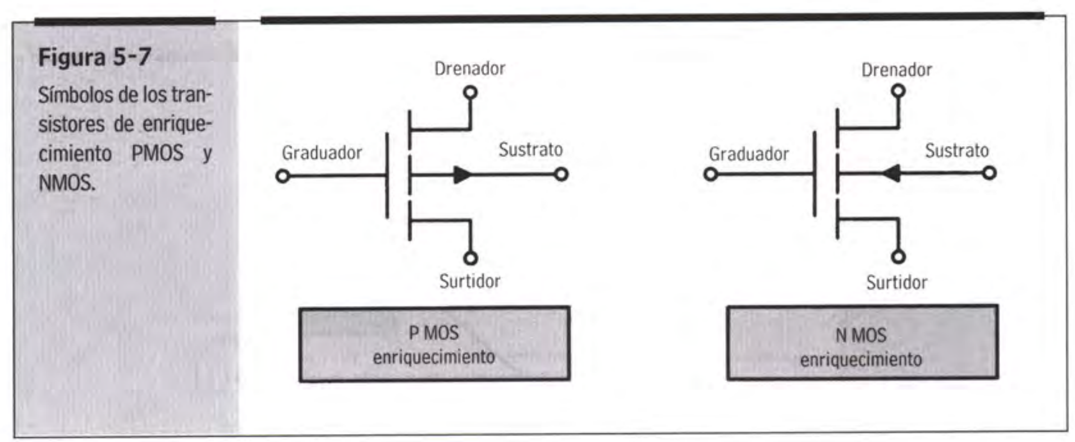


Figura 5-7
Símbolos de los transistores de enriquecimiento PMOS y NMOS.

Al mandar a masa el sustrato y el surtidor, y polarizar negativamente al drenador, en ausencia de tensión en el graduador se produce una corriente ID a través del canal P. Cuando al graduador se le aplica una tensión cada vez más positiva, los huecos del canal P son repelidos y disminuyen la anchura del canal y la corriente que circula por él. Figura 5-8.

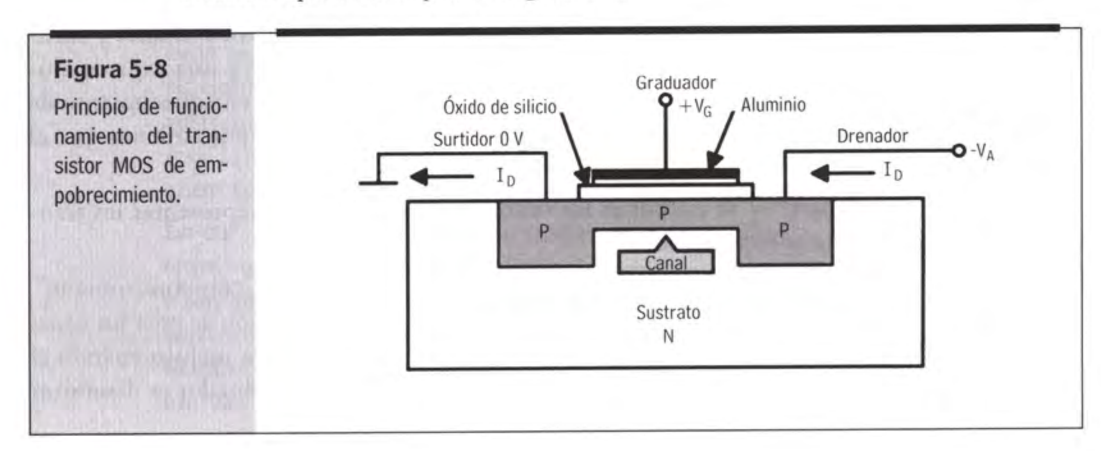


Figura 5-8
Principio de funcionamiento del transistor MOS de empobrecimiento.

Los símbolos usados para los transistores de empobrecimiento se muestran en la figura 5-9.

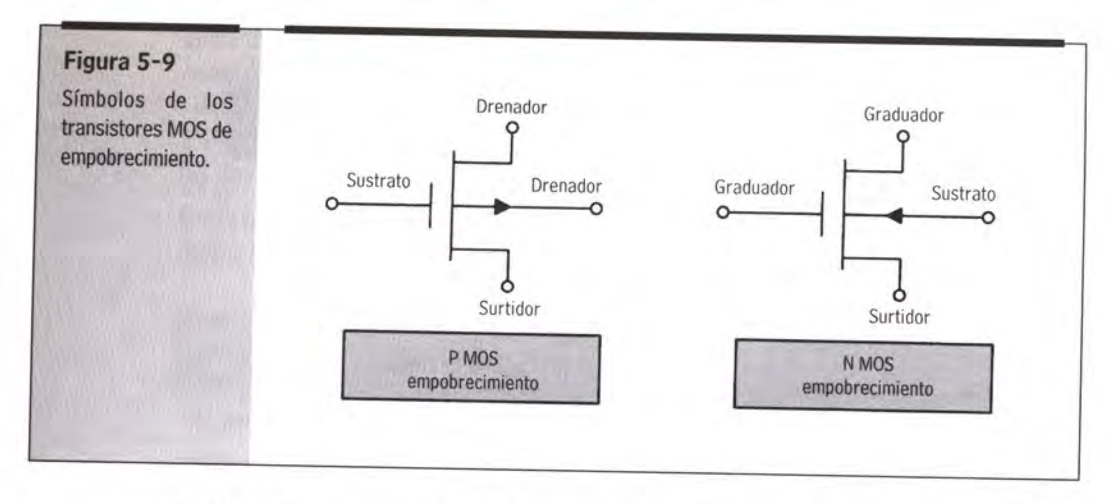


Figura 5-9
Símbolos de los transistores MOS de empobrecimiento.

Existen CI que integran en el mismo chip transistores NMOS y PMOS y que reciben el nombre de "MOS complementarios" o CMOS. Consumen menos y necesitan una sola tensión para su funcionamiento. En la figura 5-10 se ofrece un esquema de un inversor o puerta NOT con tecnología CMOS.

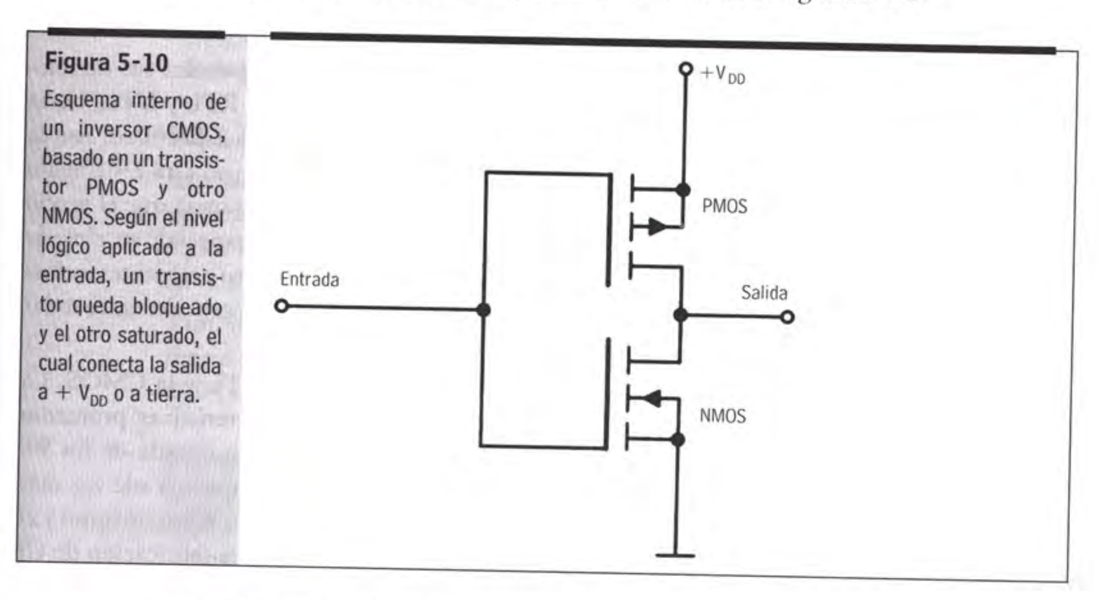


Figura 5-10
Esquema interno de un inversor CMOS, basado en un transistor PMOS y otro NMOS. Según el nivel lógico aplicado a la entrada, un transistor queda bloqueado y el otro saturado, el cual conecta la salida a $+V_{DD}$ o a tierra.

5.2. Familias lógicas

Se llama "familia lógica" a un conjunto de diversos CI, que, al ser **compatibles** entre sí, pueden utilizarse en la implementación de un circuito digital. La compatibilidad entre CI significa que se alimentan con el mismo voltaje y que las salidas de unos pueden conectarse directamente a las entradas de otros.

El impresionante desarrollo de estas familias se ha basado en los avances tecnológicos y en la feroz competencia entre sus fabricantes.

En las siguientes tablas se citan las principales familias lógicas.

FAMILIAS LÓGICAS BIPOLARES	
NOMBRE	**TIPO DE LÓGICA EMPLEADA**
RTL	Resistencia-Transistor
DTL	Diodo-Transistor
TTL	Transistor-Transistor
ECL	Acoplo por emisor
HTL	Alto umbral de ruido
I^2L	Inyección-Integrada

FAMILIAS LÓGICAS UNIPOLARES	
NOMBRE	**TIPO MOS**
NMOS	Metal-Óxido-Semiconductor Canal N
PMOS	Metal-Óxido-Semiconductor Canal P
CMOS	MOS complementarios

Hay familias lógicas que ya no se emplean, al haber sido superadas sus características por otras más modernas. Tal es el caso de las familias RTL y DTL. Otras familias sólo tienen una aplicación muy concreta en campos específicos. Así, la ECL es la familia de elevada velocidad por excelencia, y su uso está restringido a las áreas en las que la velocidad tiene mucha mayor importancia que el precio o el consumo. La familia HTL responde a un propósito especial: su elevada inmunidad al ruido, por lo que tiene muy buena aceptación en ambientes industriales hostiles. La familia I^2L es la de mayor densidad de integración entre todas las familias bipolares, pero es más lenta que la TTL.

Las dos familias que más se utilizan actualmente son la TTL y la CMOS. La primera (con muchas subfamilias) dispone de unas características promedio muy buenas y su uso ha crecido exponencialmente hasta la década de los 90. Desde hace poco tiempo, la familia CMOS y derivadas se emplean cada vez más para la construcción de memorias y microprocesadores por su bajo consumo y el reducido volumen que les acompaña. Es muy adecuada en la fabricación de CI muy complejos con circuitos repetitivos.

5.2.1. Familia TTL

Es una de las más populares y extendidas a nivel mundial, y su aceptación ha sido tan grande que de ella se derivan numerosas **subfamilias** con las que se ha pretendido mejorar alguno de los parámetros de la familia TTL estándar.

Una de las nomenclaturas más usadas, la de **Texas Instruments**, identifica los diversos modelos de la familia TTL estándar por el prefijo "74", referido a los CI que pueden trabajar con el margen de temperatura comercial, comprendido entre 0 y 70 °C. Los CI que funcionan con el rango militar de temperaturas, que comprende desde -55° a +125 °C, llevan el prefijo "54". Así, el CI 7400 contiene cuatro puertas NAND fabricadas con tecnología bipolar, propia de la familia TTL. Las dos primeras cifras indican la familia y el margen de temperaturas y las dos últimas (00) el circuito funcional interno.

En la tabla siguiente se citan las subfamilias TTL más representativas con los prefijos que las diferencian en su nomenclatura y las características más importantes.

SUBFAMILIAS TTL			
DENOMINACIÓN	PREFIJO	CARACTERÍSTICAS RELEVANTES	CI EJEMPLO
TTL ESTÁNDAR	74	PROMEDIO BUENO	7400
TTL BAJA POTENCIA	74L	BAJO CONSUMO	74L00
TTL ALTA VELOCIDAD	74H	ALTA VELOCIDAD	74H00
TTL SCHOTTKY	74S	MEJOR PROMEDIO GENERAL QUE TTL	74S00
TTL SCHOTTKY BAJA POTENCIA	74LS	SCHOTTKY DE BAJA POTENCIA	74LS00
TTL SCHOTTKY AVANZADO	74AS	MEJOR PROMEDIO VELOCIDAD-POTENCIA	74AS00
TTL SCHOTTKY AVANZADO DE BAJA POTENCIA	74ALS	GRAN REDUCCIÓN DE CONSUMO	74ALS00

5.2.2. Familia CMOS

Esta familia de CI se caracteriza porque sus circuitos internos están construidos con transistores PMOS y NMOS dentro del mismo chip. Sus excelentes cualidades han conseguido que, junto con la familia TTL, sean las dos más usadas en la mayoría de las aplicaciones.

También la familia CMOS dispone de varias subfamilias según el objetivo prioritario al que se haya enfocado su diseño. En la siguiente tabla se presentan las series más importantes de esta familia.

SERIES CMOS			
DENOMINACIÓN	PREFIJO	CARACTERÍSTICAS RELEVANTES	CI EJEMPLO
4000 / 14000	40XX/140XX	SERIE INICIAL POPULAR	4011 / 14011 4 PUERTAS NAND/2
4000B	40XXB	SERIE 4000 CON BUFFERS	4011B
74C00	74CXX	FUNCIONES Y PATILLAJE COMPATIBLE TTL	74C00
74HC00	74HCXX	ALTA VELOCIDAD	74HC00
74HCT00	74HCTXX	COMPATIBLE EN VOLTAJE CON TTL	74HCT00

Las familias 74C, 74HC y 74HCT disponen de elementos compatibles, patita a patita, con los de las subfamilias TTL. Incluso la serie 74HCT es compatible eléctricamente.

5.2.3. Análisis comparativo de características

Cada familia lógica y cada subfamilia se han diseñado con una finalidad concreta, lo que conlleva ventajas e inconvenientes. Por lo general, la mejora de una característica se consigue a costa del perjuicio que se ocasiona en otra.

Como el elemento básico con el que se construyen los circuitos de cada familia es diferente, varían los circuitos básicos fundamentales y son distintas las características de funcionamiento de los componentes que constituyen cada familia o subfamilia. La idoneidad de cada familia para su aplicación en un determinado proyecto depende de sus características. Dos ejemplos pueden aclarar estas ideas. Los circuitos digitales que deben operar en un computador, deben hacerlo a gran velocidad, lo que significa, en el caso de las puertas lógicas, que su salida debe adquirir el nivel lógico adecuado en un tiempo reducidísimo desde el instante en que se modifica el estado de las entradas. Por el contrario, en un equipo que deba estar funcionando durante largo tiempo alimentado por baterías, el factor más importante que determinará la selección de la familia idónea será el bajo consumo.

Para definir sus productos, los fabricantes de CI emplean algunas características comunes, que recogen en sus catálogos, publicaciones y hojas de referencias técnicas. A continuación se exponen los parámetros más significativos, de los que se analizan y comparan los valores usuales en las dos familias más importantes, la TTL y la CMOS.

Alimentación

Para que funcione un CI hay que aplicar a las dos patitas correspondientes la tensión de corriente continua que necesita.

En la familia TTL se designa con $\mathbf{V_{CC}}$ la patita del CI por la que se introduce el polo positivo del voltaje de c.c., que tiene un valor de +5 V (con una tolerancia permisible del 5%) en los modelos con prefijo 74. La serie militar (prefijo 54) admite una tolerancia del 10%.

El voltaje de c.c. propio de los CI de la familia CMOS se denomina $\mathbf{V_{DD}}$ y puede tener un valor comprendido entre +3 y +18 V, aunque normalmente se usen +5 V para trabajar con la misma tensión que la familia TTL.

Retardo de propagación

Es el retraso producido en la respuesta de un circuito digital. Es una medida de la rapidez con la que una puerta o un flip-flop proporcionan el valor correcto en su salida al aplicarles un estímulo en la entrada.

La **velocidad** es un parámetro dependiente del retardo y mide la frecuencia máxima con la que una puerta o un flip-flop puede cambiar de estado sin cometer errores.


BIBLIOTECA CADIZ PUBLICA

El retardo se mide en nanosegundos (ns) y la velocidad en megahercios (MHz).

Potencia de disipación

Mide la potencia consumida por una puerta y de ella depende la temperatura que alcanza el componente. Se mide en milivatios (mW).

La suma de las potencias de todos los elementos de un circuito determina el consumo total, que se utiliza para fijar la potencia de alimentación y la refrigeración adecuadas.

Producto velocidad-potencia

Los dos factores predominantes y antagónicos de una familia lógica son la velocidad y la potencia.

"*A mayor velocidad, más consumo de potencia.*" De esta regla general se desprende que un parámetro muy útil en la valoración de la idoneidad de un CI es el producto de su retardo de propagación, medido en ns, por la potencia, medida en mW. Dicho producto se mide en picojulios (pJ).

Fan-out o capacidad de carga

Es un parámetro que indica el número máximo de salidas correspondientes a elementos estándar de la misma familia lógica, que pueden conectarse a la entrada de otro.

Se reserva el nombre de "**fan-in**" o "**abanico de entrada**" al máximo número de entradas que pueden aplicarse a la salida de un elemento de una familia.

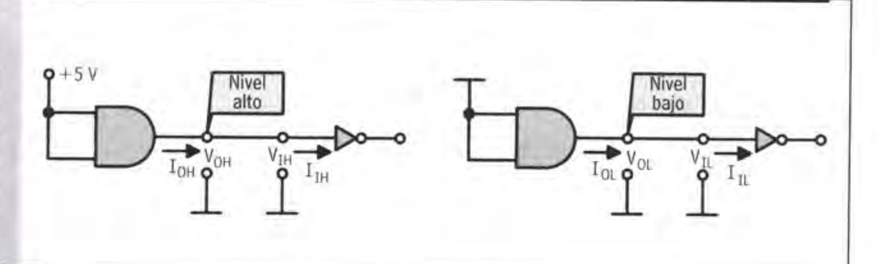
Voltajes y corrientes

Normalmente, la salida de un elemento lógico está conectada a la entrada de otro de la misma familia. Hay que tener en cuenta que, según la salida del primer elemento proporcione un nivel lógico alto o bajo, los rangos de tensión y corriente que determinan los niveles lógicos serán diferentes.

En la figura 5-11 se proponen dos esquemas de conexionado entre dos elementos lógicos, en uno de los cuales se transmite un nivel lógico alto, y en el otro, un nivel bajo. También se indican las abreviaturas empleadas para designar las tensiones y corrientes en ambos casos.

Figura 5-11

Denominaciones de voltajes y corrientes entre dos puertas, según sea alto o bajo el estado de la salida de la primera.



Cada uno de los parámetros representados en los esquemas de la figura 5-11 tiene el siguiente significado:

V_{OH} (mín): Voltaje mínimo que produce la salida de un elemento lógico cuando genera un nivel alto.

V_{OL} (máx): Máximo voltaje que puede producir en su salida un elemento lógico para el nivel bajo.

V_{IH} (mín): Voltaje mínimo para el nivel alto que puede aplicarse a una entrada de un elemento lógico.

V_{IL} (máx): Voltaje máximo para un nivel bajo aplicado a la entrada de un elemento lógico.

I_{OH}: Corriente de salida que proporciona un elemento lógico que proporciona nivel alto. En la familia TTL, I_{OH} (máx) = 0, 4 mA.

I_{IH}: Corriente de entrada de un elemento lógico cuando se aplica un nivel alto. En la familia TTL, I_{IH} (máx) = 40 μA.

I_{OL}: Corriente de salida con nivel bajo. En TTL, I_{OL} (máx) = 16 mA.

I_{IL}: Corriente de entrada con nivel bajo. En TTL, I_{IL} (máx) = -1, 6 mA.

En el caso de la familia TTL, los valores de los voltajes para los niveles alto y bajo (según se trate de entrada o salida) se especifican en los gráficos de la figura 5-12.

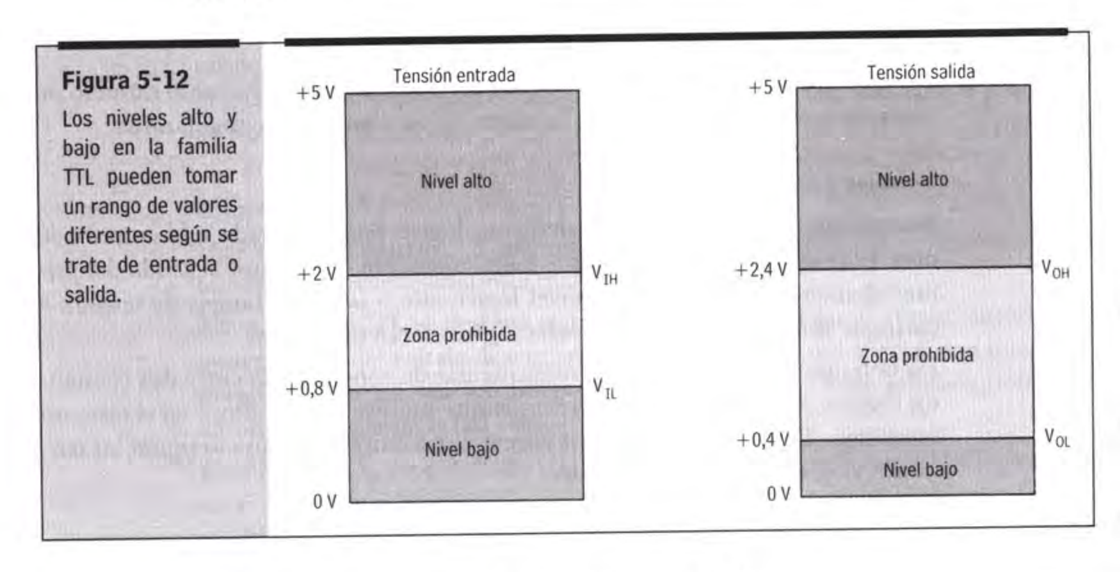


Figura 5-12
Los niveles alto y bajo en la familia TTL pueden tomar un rango de valores diferentes según se trate de entrada o salida.

En el caso de la familia CMOS, si V_{DD} = + 5 V,

V_{OL}(máx) = 0 V

V_{OH}(mín) = +5 V

V_{IL}(máx) = +1,5 V (30% de V_{DD})

V_{IH}(mín) = +3,5 V (70% de V_{DD})

Inmunidad al ruido

Es la cantidad de ruido o voltaje indeseado, expresado en V o mV, que puede superponerse a un estado lógico (alto o bajo) sin que cambie incorrectamente. Así, por ejemplo, un nivel lógico bajo aplicado a una entrada de una puerta lógica puede transformarse en nivel alto si al voltaje correspondiente al nivel bajo se le añade otro extraño que le eleve lo necesario. El parámetro de "inmunidad al ruido" mide el voltaje máximo que puede sumarse a un nivel lógico sin hacerle cambiar.

De acuerdo con la figura 5-12, si la salida de un elemento lógico proporciona nivel alto (V_{OH}), como mínimo el voltaje proporcionado será de 2,4 V. Como resulta que el voltaje a partir del cual se considera nivel alto en una entrada es de 2 V, se puede soportar un ruido externo de hasta -0,4 V sin que exista cambio de nivel lógico entre la salida y la entrada de dichos componentes. En este caso, que es el que corresponde a la familia TTL, la inmunidad al ruido es de 0,4 V.

La familia CMOS dispone de una inmunidad al ruido muy superior a la TTL.

Entradas no usadas

Existe una diferencia notable en el comportamiento de elementos TTL y CMOS cuando en ellos se deja sin conectar alguna de sus entradas. Así, una entrada desconectada o al aire en un componente de la familia TTL se comporta como si dicha entrada tuviera nivel lógico alto. Por ejemplo, en una puerta OR de tres entradas, si una de ellas está desconectada es como si estuviese recibiendo un nivel alto, por lo que la salida de dicha puerta siempre proporcionará un nivel alto. Figura 5-13.

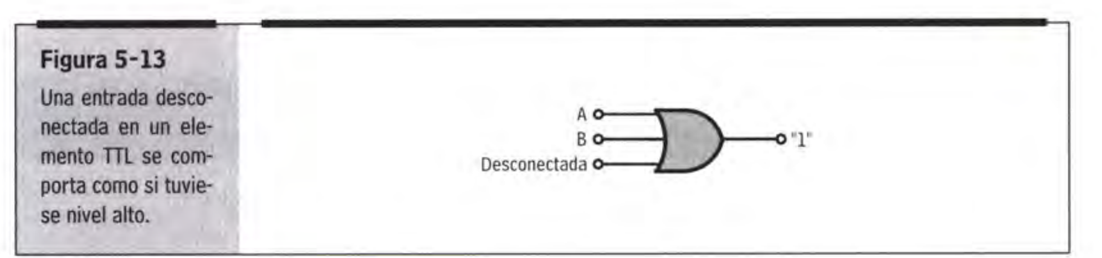


Figura 5-13
Una entrada desconectada en un elemento TTL se comporta como si tuviese nivel alto.

Para evitar la acción indeseada de la entrada desconectada en la puerta OR de la figura 5-13, habría que mandar a tierra dicha entrada como se refleja en la figura 5-14. La salida de la OR sólo depende de los niveles de las entradas A y B.

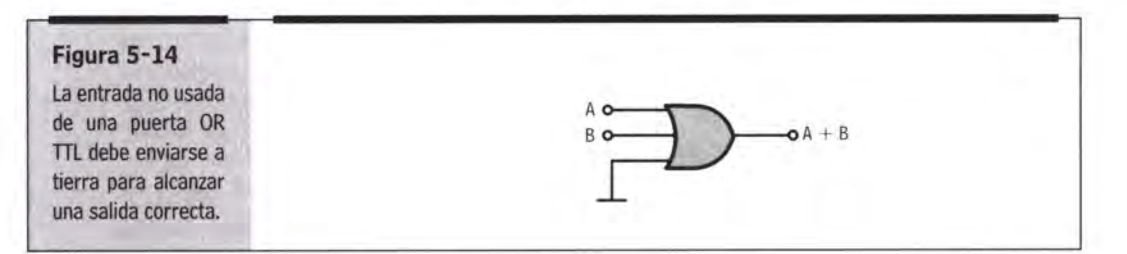


Figura 5-14
La entrada no usada de una puerta OR TTL debe enviarse a tierra para alcanzar una salida correcta.

Incluso en aquellos casos en que las entradas con nivel alto no repercuten en la salida (como en la puerta AND) debe evitarse dejar al aire entradas de dispositivos lógicos puesto que actúan como antenas que captan ruidos y pueden ocasionar fallos de funcionamiento. En la figura 5-15 se ofrecen algunas soluciones para no dejar sin conectar una entrada de una puerta AND.

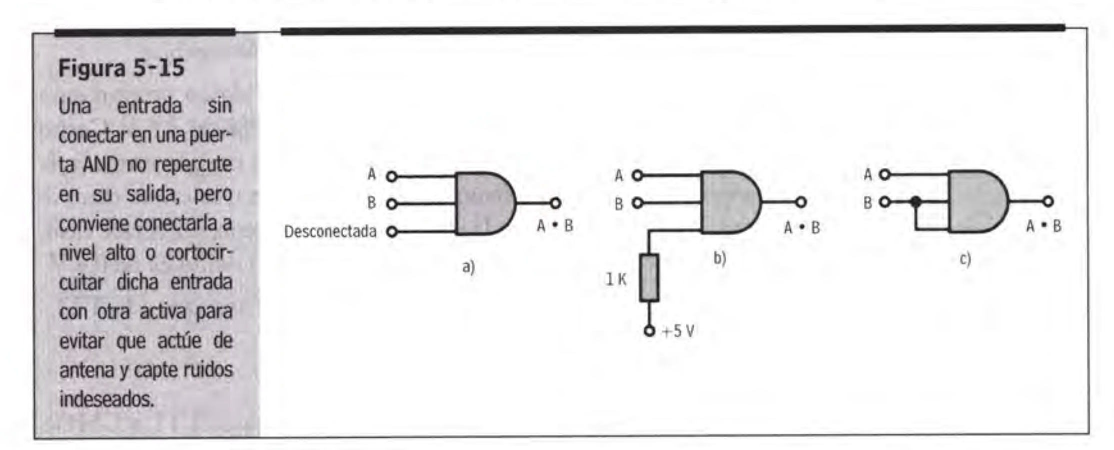


Figura 5-15
Una entrada sin conectar en una puerta AND no repercute en su salida, pero conviene conectarla a nivel alto o cortocircuitar dicha entrada con otra activa para evitar que actúe de antena y capte ruidos indeseados.

En ninguna circunstancia deben quedar sin conectar entradas de elementos de la familia CMOS, porque a través de ellas podrían captarse ruidos y cargas estáticas que podrían destruir el canal N o P del transistor de efecto de campo al que pertenecen.

En la siguiente tabla se exponen las características relevantes de las subfamilias TTL y de las dos series CMOS más populares.

	TTL (V_{CC} = 5 V)					CMOS (V_{DD} = 5 V)	
SUBFAMILIA	**74**	**74S**	**74AS**	**74LS**	**74ALS**	**4000B**	**74HC**
RETARDO DE PROPAGACIÓN (ns)	9	3	1,7	9,5	4	50	8
POTENCIA CONSUMIDA (mW) FUNCIONAMIENTO ESTÁTICO	10	20	8	2	1,2	$1 \cdot 10^{-3}$	$2,5 \cdot 10^{-3}$
PRODUCTO VELOCIDAD-POTENCIA A 100 kHz (pJ)	90	60	13,6	19	4,8	5	1,4
VELOCIDAD MÁXIMA (MHz)	35	12,5	200	45	70	12	40
INMUNIDAD AL RUIDO (V)	0,4	0,3	0,3	0,3	0,4	1,5	0,9
V_{IH} (mín) (V)	2		2	2	2	3,5	3,5

(Continuación)	TTL (V_{CC} = 5 V)					CMOS (V_{DD} = 5 V)	
SUBFAMILIA	74	74S	74AS	74LS	74ALS	4000B	74HC
V_{IL} (máx) (V)	0,8		0,8	0,8	0,8	1,5	1
V_{OH} (mín) (V)	2,4		2,7	2,7	2,7	4,95	4,9
V_{OL} (máx) (V)	0,4		0,5	0,5	0,4	0,05	0,1
I_{IH} (máx) (μA)	40		200	20	20	1	1
I_{IL} (máx) (mA)	1,6		2	0,4	0,1	0,001	0,001
I_{OH} (máx) (mA)	0,4		2	0,4	0,4	0,4	4
I_{OL} (máx) (mA)	16		20	8	8	0,4	4
Fan-out	10	25	50	10	10	> 50	> 100

Conclusiones

1º) En aplicaciones de tipo general, se puede considerar a la familia TTL la más empleada, pero la CMOS está cada vez a menos distancia.

2º) Otras familias distintas a la TTL y la CMOS sólo se usan en aplicaciones muy específicas. Tal es el caso de la familia ECL, que se aplica en circuitos computacionales de elevada velocidad, y el de la familia HTL, que tiene su campo de aplicación en ambientes industriales hostiles con un alto nivel de ruidos y perturbaciones electromagnéticas.

3º) La familia TTL, y en especial alguna de sus subfamilias, aventaja a la CMOS en "velocidad".

4º) Las características más sobresalientes de la familia CMOS son:

a) Bajo consumo.

b) Elevada inmunidad al ruido.

c) Amplio margen en la tensión de alimentación $+V_{DD}$.

d) Capacidad de carga o fan-out muy alto.

Los CI de la familia CMOS exigen más cuidado en su manipulación práctica, no debiéndose tocar directamente con las manos, pues se corre el riesgo de destruirlos al aplicar la tensión estática a las patitas del encapsulado, ya que, como se ha indicado, algunas de ellas están conectadas a los canales de los transistores de efecto de campo. Tampoco debe dejarse sin conectar ninguna de las entradas de los dispositivos lógicos, y al realizar la soldadura de su patillaje es conveniente tomar precauciones especiales para evitar la transmisión de calor al circuito electrónico interno.

Ejemplo 5-1

Analizando las características de las subfamilias TTL y las series CMOS, responder a las siguientes preguntas:

a) ¿Cuál de ellas es la más veloz?

b) ¿Cuál tiene menos consumo?

c) ¿Cuál posee la mejor relación velocidad-consumo?

d) ¿Cuál dispone de la mayor inmunidad al ruido?

e) ¿Cuál proporciona la máxima corriente de salida para el nivel lógico bajo?

SOLUCIÓN

a) 74AS. b) 4000B. c) 74HC. d) 4000B. e) 74AS.

5.3. Encapsulados y nomenclatura de CI

La empresa americana **Texas Instruments**, una de las mayores fabricantes de CI, clasifica a éstos atendiendo al grado de complejidad interna. De esta forma, según el número de puertas, o su equivalente, que contiene internamente el chip, los agrupa en cuatro categorías:

1º) **SSI: Pequeña escala de integración.**
En este nivel se incluyen los CI con un máximo de 12 puertas lógicas.

2º) **MSI: Mediana escala de integración.**
Los CI de esta categoría contienen de 12 a 100 puertas.

3º) **LSI: Alta escala de integración.**
Contienen de 100 a 1.000 puertas.

4º) **VLSI: Muy alta escala de integración.**
Más de 1.000 puertas o complejidad equivalente.

Cuando el circuito electrónico interno de un chip no está formado a base de puertas, el tipo al que pertenece se determina por la complejidad equivalente referida al número de transistores u otros elementos característicos.

Encapsulado

Hay cuatro tipos básicos de encapsulamientos:

1°. Encapsulado de doble fila.

Recibe el nombre de **DIP** (*Dual-in-line Package*) y es uno de los más usados. La cápsula, construida generalmente con material plástico o cerámico, dispone de dos hileras de patitas que se insertan en los zócalos normalizados y placas de circuito impreso.

Las patitas se distinguen por un número que se asigna ordenadamente desde el 1 al último, siguiendo el sentido de las agujas del reloj y comenzando por la patita situada a la izquierda de una marca o referencia, como se muestra en la figura 5-16.

Figura 5-16
Numeración de las patitas de un circuito integrado TTL con cápsula DIP y diagrama de conexiones de sus patitas.

El número de patitas de las cápsulas DIP está normalizado, existiendo modelos con 14, 16, 18, 20, 24, 40, 64, etc., patitas.

2°. Encapsulado plano o flat-pack.

Suelen ser de material cerámico y se caracterizan por ocupar un volumen muy reducido. Sus patitas se hallan estiradas para permitir su soldadura de forma automática.

3°. Encapsulado cilíndrico.

Suele emplearse material metálico y la forma externa es cilíndrica, similar a la de algunos modelos de transistores. Su mayor inconveniente es que admite un número muy reducido de patitas.

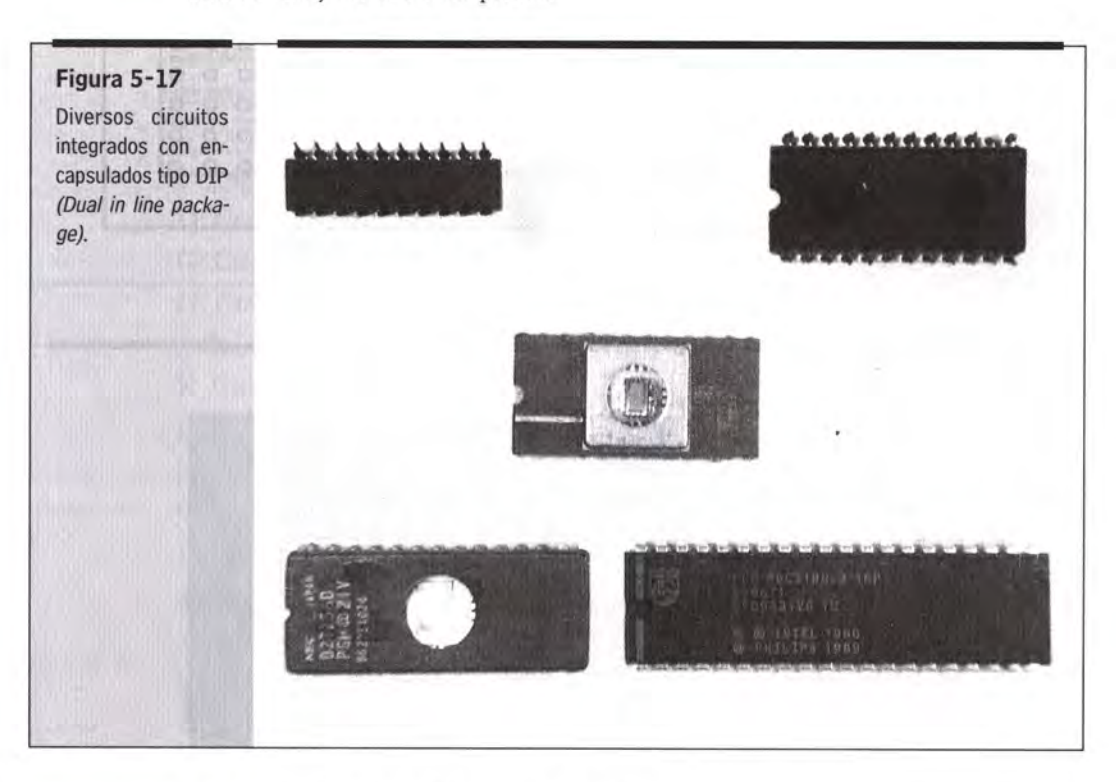


Figura 5-17
Diversos circuitos integrados con encapsulados tipo DIP *(Dual in line package).*

4°. Encapsulado de matriz o rejilla de patitas.

Cuando el número de patitas es muy elevado, se distribuyen a lo largo del perímetro de la cápsula en varias hileras. A esta disposición del patillaje se le denomina PGA (*Pin Grid Array*) o "rejilla de patitas".

En la figura 5-18 se muestra la cápsula con la distribución de sus 168 patitas, correspondiente al microprocesador 486 de **Intel**. La denominación de cada patita utiliza un número y una letra para referenciar la fila y la columna en que se encuentra.

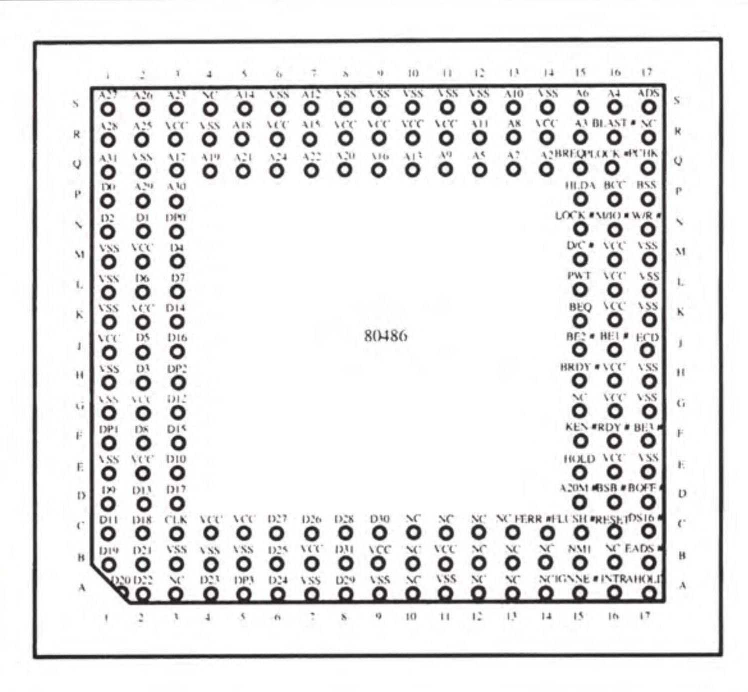


Figura 5-18

Diagrama de conexionado del 486 visto el encapsulado por el lado de las patitas.

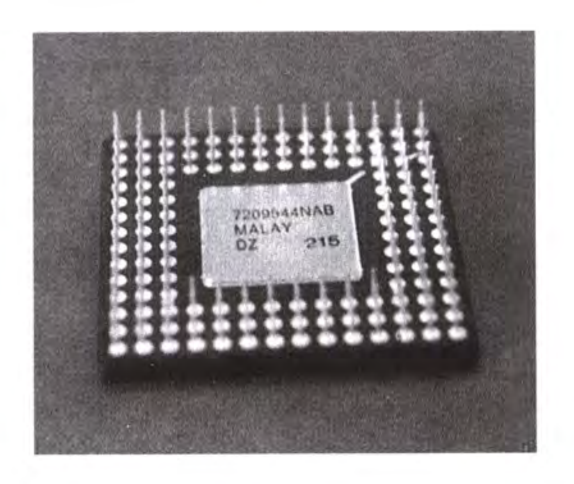


Figura 5-19

Fotografía de un circuito integrado con encapsulado de matriz de patitas PGA (*Pin Grid Array*).

5.3.1. Nomenclatura

No existe unificación en la denominación de los Cl, empleando cada fabricante una nomenclatura particular. A continuación se describen las nomenclaturas más representativas.

Grupo Proelectrón

La mayoría de los fabricantes de CI europeos utilizan este código, que emplea en la designación tres letras y tres números, cuyo significado es el siguiente:

Primera letra: indica el tipo de CI de que se trata, de acuerdo con el siguiente código:

T: CI lineal o analógico (no digital).

F: CI digital que forma familia con otros.

S: CI digital que no forma familia con otros.

U: CI mixto, es decir, analógico-digital.

Segunda letra: todos los CI que comparten la misma segunda letra pertenecen a la misma familia. La letra C se corresponde con la familia DTL, la D, con la MOS y la J, con la TTL.

Tercera letra: indica la función del circuito interno.

A: Amplificación lineal.

B: Demodulación.

C: Oscilación.

D: Combinación de circuitos lineales.

G: Circuitos híbridos.

H: Circuitos combinacionales.

J: Circuitos multiestables.

K: Circuitos monoestables.

L: Convertidores de nivel digital.

N: Multiestables. Función memoria de media o alta escala de integración.

Q: Circuitos de memoria de lectura-escritura.

R: Circuitos de memoria de sólo lectura.

S: Amplificador de lectura.

Y: Varios.

Cifra de tres números. Los dos primeros se corresponden con el número de serie y el tercero refleja el margen de temperaturas de acuerdo con la siguiente relación:

1: de 0 a 70 °C.

2: de -55 a +125 °C.

3: de -10 a +85 °C.

4: de 15 a 55 °C.

5: de -40 a +75 °C.

6: de -40 a +85 °C.

0: otros márgenes.

Ejemplo 5-2

Indicar las características que se pueden determinar para las dos siguientes denominaciones de CI:

a) FJH 111.

b) TAA522.

SOLUCIÓN.

a) F: CI digital que forma familia con otros.
J: Perteneciente a la familia TTL.
H: Función lógica correspondiente a un circuito combinacional. En este caso se trata de dos puertas NAND de cuatro entradas.
11: Número de serie.
1: Margen de temperatura de 0 a 70 °C.

b) T: CI lineal o analógico.
A: Pertenece a una familia.
A: Amplificación lineal.
52: Número de serie.
2: Margen de temperatura entre -55 y +125 °C.

Nomenclatura de Texas Instruments Inc para la serie TTL.

En 1964, **Texas Instruments** inició la fabricación de algunos modelos de CI simples de la familia TTL para aplicaciones militares. Se trataba de la serie 54, que puede funcionar con un margen de temperatura entre -55 y +125 °C. Poco después, tras el éxito obtenido, comercializó la serie 74 para aplicaciones generales y de bajo precio, teniendo restringido el margen de temperatura de trabajo entre 0 y 70 °C.

La gran aceptación y las enormes ventas de la serie TTL estándar 54/74 dio origen a bastantes subfamilias, que se diseñaban para mejorar algún parámetro determinado. Para distinguir los componentes de esta familia, **Texas** utiliza una nomenclatura que se compone de dos letras, cinco números y una letra final, que se interpreta de la siguiente forma:

a) *Primeras dos letras:* **SN (Semiconductor Network)**, propias de este fabricante.

b) *Dos números siguientes:* especifican el margen de temperatura de trabajo con la siguiente correspondencia:
72, 74 y 75: margen comercial de 0 a 70 °C.
52, 54 y 55: margen militar de -55 a +125 °C.

c) *Tres números siguientes:* determinan la función digital que desarrollan (tipo de puerta, contador, etc.). A veces, este campo sólo consta de dos números.

d) *Última letra:* indica el tipo de encapsulado.
J: DIP cerámico.
N: DIP plástico.
H, U, T, W, Z: encapsulado plano o *flat-pack*.
L: cápsula cilíndrica de metal.

Ejemplo 5-3

Indicar las características que se desprenden de un CI cuya nomenclatura es: SN 7400J.

SOLUCIÓN

1º SN significa que el fabricante es **Texas Instruments.**

2º 74 indica que se trata de un CI digital de la familia TTL estándar con margen de temperatura comercial.

3º 00 es el número de serie que determina que el CI consta de cuatro puertas NAND de dos entradas.

4º La J indica que el CI está encapsulado con el formato DIP cerámico.

En la siguiente tabla se indican los principales modelos de CI conteniendo puertas y elementos lógicos simples SSI, de la familia TTL estándar SN 54/74.

SERIE SN 54/74	FAMILIA TTL ESTÁNDAR
PUERTAS LÓGICAS SSI	
MODELO	**FUNCIÓN**
54/7400	4 puertas NAND de dos entradas.
54/7401	4 puertas NAND de dos entradas, con colector abierto.
54/7402	4 puertas NOR de dos entradas.
54/7403	4 puertas NAND de dos entradas, con colector abierto.
54/7404	6 inversores o puertas NOT.
54/7406	6 inversores con amplificación y colector abierto.
54/7407	6 amplificadores o buffers con colector abierto.
54/7408	4 puertas AND de dos entradas.
54/7409	4 puertas AND de dos entradas y colector abierto.
54/7410	3 puertas AND de tres entradas.
54/7412	3 puertas AND de tres entradas y colector abierto.
54/7413	1 circuito de disparo "Schmitt-Trigger" de cuatro entradas.
54/7414	6 inversores circuitos de disparo.
54/7416	6 buffers amplificadores, inversores.
54/7417	6 buffers amplificadores, con colector abierto.
54/7420	2 puertas NAND de cuatro entradas.
54/7425	2 puertas NAND de cuatro entradas con señal de Strobe.
54/7426	4 puertas NAND de dos entradas y colector abierto.
54/7427	3 puertas NOR, de tres entradas.
54/7430	1 puerta NAND de ocho entradas.
54/7432	4 puertas OR de dos entradas.
54/7440	2 puertas NAND de cuatro entradas con buffer.
54/7450	2 puertas AND-OR-INVERSOR de 2 x 2 entradas.
54/7451	2 puertas AND-OR-INVERSOR de 2 x 2 entradas.
54/7453	1 puerta AND-OR-INVERSOR cuádruple de dos entradas.
54/7486	4 puertas EOR de dos entradas.
54/74125	4 buffers no inversores triestado con ENABLE negada.
54/74126	4 buffers no inversores.

Aunque cada fabricante de CI digitales utiliza en su denominación su propia nomenclatura, una de las más empleadas es la de **Texas** y el número de serie, que

referencia a los dos o tres números finales e identifica la función interna, es el mismo en la mayoría de las nomenclaturas. Así, por ejemplo, al CI que **Texas** denomina **SN 7400 N**, la empresa **National** le llama DM 7400N, **Motorola** MC 7400P, **Fairchild** 9N00 y **Siemens** FLH101. Cada fabricante dispone de un catálogo con sus productos en el que se indican sus características técnicas y los modelos equivalentes de otras marcas.

Ejemplo 5-4

Dado el esquema de la figura 5-20, indicar qué tipo de modelos de CI y en qué cantidad se precisan para implementarlo. Los CI deben ser de la familia TTL estándar y su nomenclatura debe corresponder a la del fabricante **Texas Instruments**.

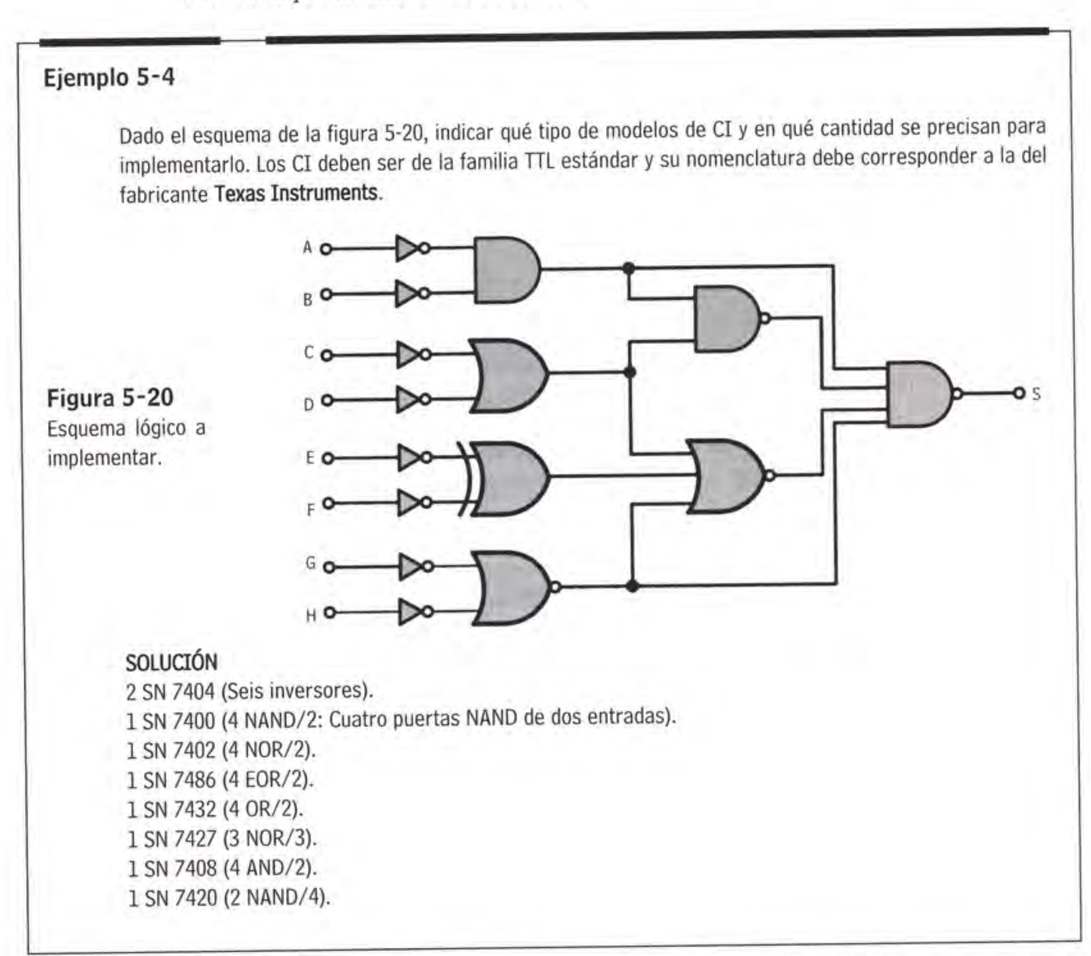


Figura 5-20
Esquema lógico a implementar.

SOLUCIÓN
2 SN 7404 (Seis inversores).
1 SN 7400 (4 NAND/2: Cuatro puertas NAND de dos entradas).
1 SN 7402 (4 NOR/2).
1 SN 7486 (4 EOR/2).
1 SN 7432 (4 OR/2).
1 SN 7427 (3 NOR/3).
1 SN 7408 (4 AND/2).
1 SN 7420 (2 NAND/4).

Respecto a los modelos más representativos de la familia CMOS, se citan los de la popular serie 4000.

4000: 2 NOR/3 y 1 inversor.

4001: 4 NOR/2.

4002: 2 NOR/4.

4011: 4 NAND/2.

4012: 2 NAND/4.

4013: Dos biestables tipo D.

4015: Dos registros de desplazamiento de 4 bits.

4017: Divisor-contador de décadas.

4020: Contador binario.

4023: 3 NAND/3.

4025: 3 NOR/3.

4027: Dos biestables J-K.

4028: Decodificador BCD/decimal.

4035: Registro desplazamiento de entrada serie/paralelo y salida paralelo.

4042: Cuatro registros D.

4044: Cuatro flip-flops R-S con NOR.

4049: Seis buffers inversores.

4051: Multiplexor/demultiplexor analógico de ocho canales.

4052: Dos multiplexores/demultiplexores de cuatro canales.

4068: 1 NAND/8.

4069: Seis inversores.

4070: 4 EOR/2.

4071: 4 OR/2.

4072: 2 OR/4.

4081: 4 AND/2.

La serie **74HCT** dispone de las funciones lógicas más empleadas de la familia TTL y se alimenta con +5 V, pudiéndose interconectar directamente componentes de las dos familias.

5.4. Elementos lógicos especiales

En el conjunto de componentes que conforman las funciones lógicas básicas de las familias, existen algunos dispositivos peculiares caracterizados por poseer propiedades concretas muy útiles en diversos circuitos digitales. Los más importantes son:

Componentes con "salidas con colector abierto".

Puertas "AND-OR-INVERSOR".

Dispositivos "triestado", con entrada de habilitación.

Circuitos de disparo, "Schmitt-Trigger".

Buffers amplificadores.

Dado el interés práctico de estos dispositivos, se realiza una breve descripción de todos ellos.

5.4.1. Componentes con "salidas con colector abierto"

En las realizaciones digitales prácticas se precisa, frecuentemente, unir las salidas de varias puertas TTL entre sí con el fin de realizar una operación "AND

cableada" sin tener que colocar una puerta AND. En la figura 5-21 se muestra la representación gráfica de la operación "AND cableada".

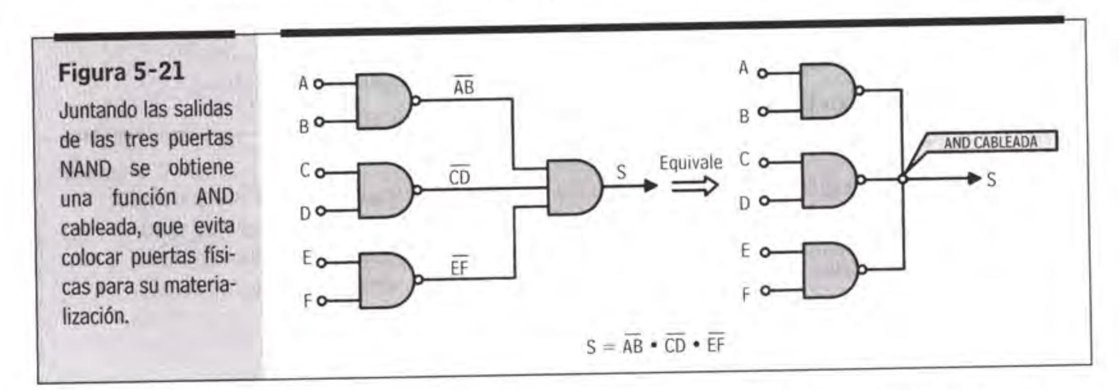


Figura 5-21
Juntando las salidas de las tres puertas NAND se obtiene una función AND cableada, que evita colocar puertas físicas para su materialización.

La unión de varias salidas entre sí se comporta de forma similar a la de una puerta AND, puesto que sólo proporcionará en su salida un nivel lógico alto cuando todas las que se hallen conectadas tengan dicho nivel. Si alguna de esas salidas posee nivel bajo, forzará el punto de unión **S** a dicho nivel, produciéndose, además, un cortocircuito desde los niveles altos de las otras salidas, ocasionando el paso de una elevada corriente que podría deteriorar alguno de los transistores internos de los CI, dejándolos inservibles. Para posibilitar la implementación de la "AND cableada", evitando el peligro de cortocircuitos, se fabrican CI especiales, denominados "con colector abierto". Estos elementos carecen en las salidas de las puertas de la resistencia que las conecta con la alimentación V_{CC}. Como se muestra en la figura 5-22, hay que colocar una resistencia exterior, R_L, entre la unión de las salidas con colector abierto y V_{CC}.

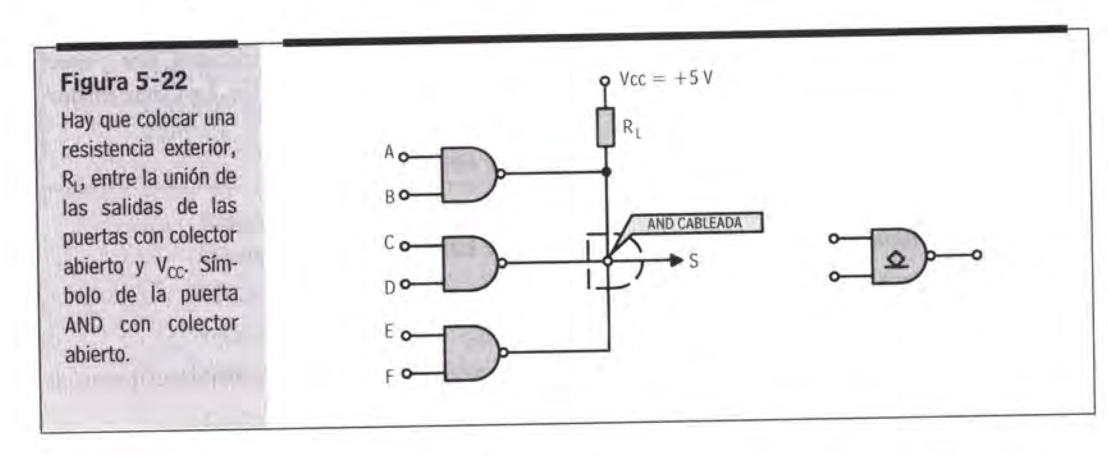
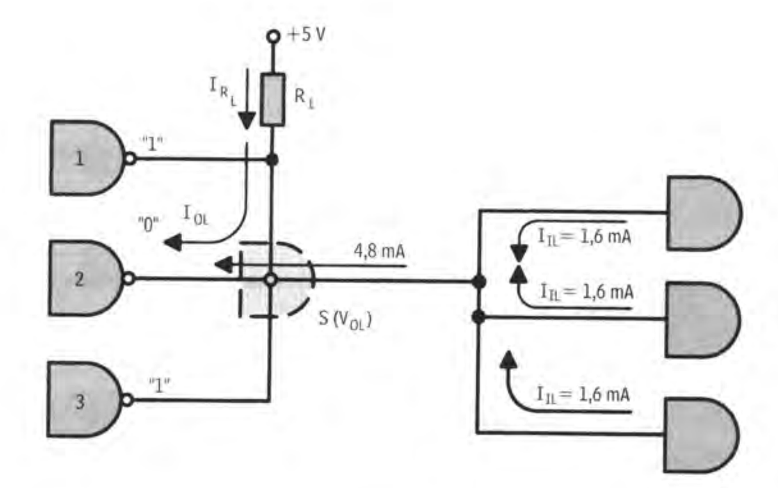


Figura 5-22
Hay que colocar una resistencia exterior, R_L, entre la unión de las salidas de las puertas con colector abierto y V_{CC}. Símbolo de la puerta AND con colector abierto.

La misión de R_L es suministrar la alimentación a las salidas de las puertas y limitar el valor de la corriente (I_{OH} e I_{OL}), garantizando que V_{OH} sea igual o mayor que 2,4 V y que V_{OL} sea igual o menor que 0,4 V.

Ejemplo 5-5

Dado el esquema de la figura 5-23, en el cual se supone que sólo la puerta 2 de la "AND cableada" proporciona nivel lógico bajo, calcular el valor de R_L si dicha unión está aplicada a tres entradas de otras tantas puertas.



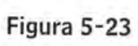


Figura 5-23

En la AND-cableada, sólo la puerta 2 tiene nivel bajo y la salida S alimenta a tres entradas.

SOLUCIÓN

Teniendo en cuenta que la $I_{OL\ máx} = 16$ mA y puesto que,

$I_{OL} = 3 \cdot I_{IL} + I_{RL}$

$I_{RL} = I_{OL} - 3 \cdot I_{IL} = 16 - 4,8 = 11,2$ mA

Como el punto de salida S tiene nivel bajo, se debe garantizar que la tensión V_{OL} sea igual o menor que 0,4 V, con lo cual:

$$R_L = \frac{V_{RL}}{I_{RL}}$$

$V_{RL} = V_{CC} - V_{OL\ máx} = 5 - 0,4 = 4,6$ V

$$R_L = \frac{4,6\ \text{V}}{11,2\ \text{mA}} = 410,7\ \Omega$$

5.4.2. Puertas AND-OR-INVERSOR

La misión de estas puertas es sustituir la función **AND cableada** evitando el empleo de puertas con colector abierto, mejorando de esta forma el tiempo de respuesta y el fan-out. En la figura 5-24 se presenta el esquema interno y el diagrama de conexiones del CI 7451, que contiene dos puertas AND-OR-INVERSOR de 2 x 2 entradas.

Una puerta AND-OR-INVERSOR implementa la función AND cableada directamente. Así, en el caso del modelo 7451, en la figura 5-25 se muestra la operación lógica que realiza.

Figura 5-24

Circuito interno y diagrama de conexiones del CI 7451.

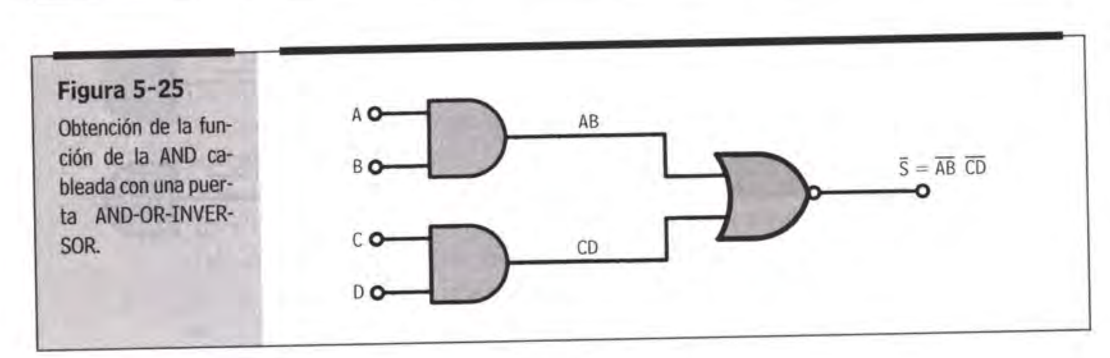


Figura 5-25

Obtención de la función de la AND cableada con una puerta AND-OR-INVERSOR.

5.4.3. Dispositivos triestado con entrada de habilitación

Un elemento muy utilizado en la estructura de los computadores es el que recibe el apelativo de **triestado,** como consecuencia de que su salida puede trabajar con tres estados posibles:

1.- **ALTO**

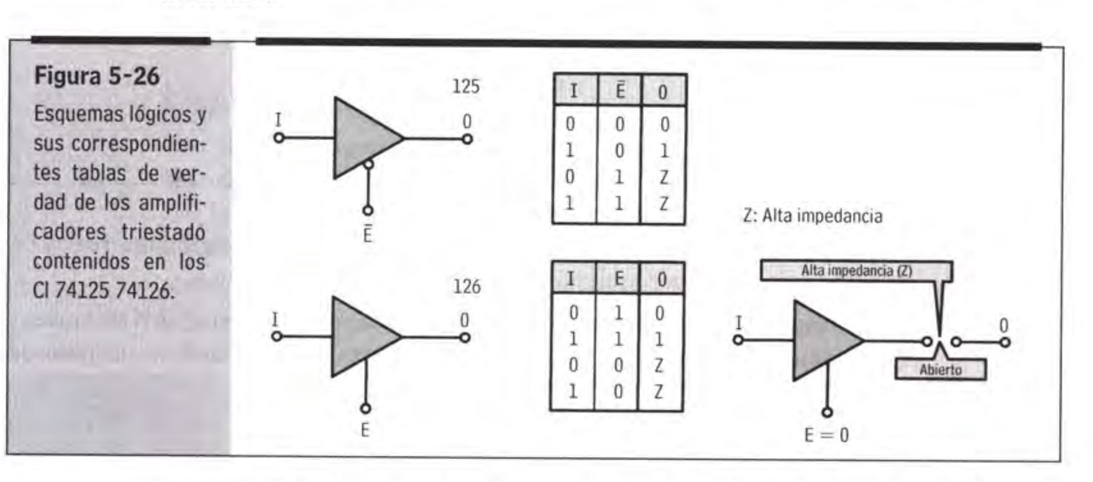
2.- **BAJO**

3.- **ALTA IMPEDANCIA**

El nuevo estado, o sea, el tercero, llamado de "alta impedancia", en la práctica se puede considerar como si la salida se comportase igual que un cable desconectado o en circuito abierto, que no guarda relación alguna con tierra y con VCC. Es como si la salida del elemento triestado quedase desconectada del circuito electrónico interno cuando adopta el estado de alta impedancia.

Los elementos triestado disponen de una entrada especial, denominada técnicamente **ENABLE**, que permite o prohíbe el funcionamiento binario normal de su salida. Dicha entrada de control (**E**), puede ser activa por nivel alto o por nivel bajo. Cuando la señal ENABLE está activa, el elemento lógico funciona normalmente y en su salida aparecen los niveles lógicos alto y bajo. En caso de estar inactiva la señal **E**, la salida del elemento triestado queda en alta impedancia.

Los CI de la familia TTL 74125 y 74126 contienen cuatro buffers o amplificadores no inversores triestado. Su esquema lógico junto a su tabla de verdad se ofrecen en la figura 5-26. El buffer o amplificador no inversor tiene la misión de obtener en su salida el mismo nivel lógico aplicado a su entrada, pero amplificado. Además, los dos modelos a los que se hace referencia tienen la capacidad de dejar su salida en alta impedancia cuando se deja inactiva la señal ENABLE. La diferencia entre el 74125 y el 74126 estriba en el nivel lógico que hace activa a la señal de control **E**, que en el primero es por nivel bajo y en el segundo por nivel alto.



I	Ē	0
0	0	0
1	0	1
0	1	Z
1	1	Z

I	E	0
0	1	0
1	1	1
0	0	Z
1	0	Z

Figura 5-26
Esquemas lógicos y sus correspondientes tablas de verdad de los amplificadores triestado contenidos en los CI 74125 74126.

La principal ventaja de los elementos triestado reside en la posibilidad de controlar los momentos en que su salida es válida. Esta característica permite alimentar a una misma línea desde varios dispositivos triestado, con el único cuidado de que no exista en cada momento más que uno de ellos con la señal ENABLE activa, lo que supone que los restantes están desconectados de la línea y no se van a producir cortocircuitos. La línea común a la que alimentan las tres puertas triestado mostradas en la figura 5-27 soportará en cada instante la información de una de las puertas, siempre que se active una sola señal ENABLE.

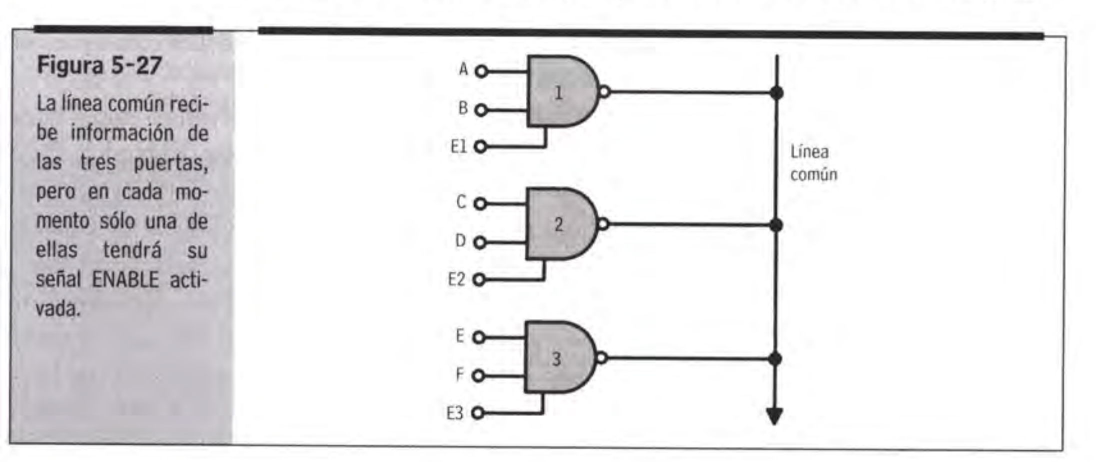


Figura 5-27
La línea común recibe información de las tres puertas, pero en cada momento sólo una de ellas tendrá su señal ENABLE activada.

Cuando en el esquema de la figura 5-28 se desea que la línea común transporte la información digital que sale de la puerta 2, se activa su correspondiente señal ENABLE (**E2 = 1**). Las otras dos puertas deben quedar desconectadas o en alta impedancia y sus señales ENABLE deberán estar desactivadas a nivel bajo.

En la figura 5-28 se presenta el símbolo particular que recomienda la normativa **IEEE/ANSI** para los elementos con salida triestado.

Figura 5-28
Representación de una puerta NAND con salida triestado según la norma IEEE/ANSI.

Respecto a la familia CMOS, conviene tener en cuenta las siguientes reglas en relación con el comportamiento de las salidas de sus elementos.

1ª) Nunca deben conectarse entre sí varias salidas de elementos CMOS convencionales. Se puede ocasionar la avería interna del CI debido a un paso de corriente excesivo.

2ª) Existen modelos especiales de dispositivos CMOS con salidas en circuito abierto que pueden cortocircuitarse entre sí, debiendo colocar una resistencia de valor adecuado a la alimentación VDD. Su comportamiento es similar a los CI TTL con colector abierto.

3ª) También existen modelos CMOS con salida triestado, que funcionan y se utilizan de la misma forma que los TTL.

5.4.4. Buffers amplificadores

Como se ha comentado en la descripción de los dispositivos triestado, estos componentes se emplean para incrementar la potencia de una señal sin afectar a su estado lógico. Es decir, son capaces de aumentar el número de elementos que pueden conectarse a su salida, tanto para el nivel lógico alto como para el bajo.

De esta forma, al situarlos en la salida de los circuitos, pueden activar cargas tales como lámparas, LED, displays de 7 segmentos, micro-relés, etc. El modelo 7407 contiene seis amplificadores con colector abierto.

Hay "buffers amplificadores inversores", que, además de amplificar la señal de entrada, la invierten. Es el caso del modelo 7406.

En la figura 5-29 se representa la salida S de un diseño digital que tiene que controlar un micro-relé que trabaja a 12 V y 20 mA. Se usa el CI 7407, que soporta una V_{OH} máxima de 15 V y una I_{OL} de 40 mA. Cuando se aplica un nivel bajo a la entrada del buffer, su salida queda a 0 V, prácticamente, y el micro-relé absorbe los 12 V de alimentación y 20 mA, activándose. Si lo que transmite y

amplifica el 7407 es nivel alto, no circula corriente por el circuito de salida del buffer y el micro-relé queda sin tensión, desactivado.

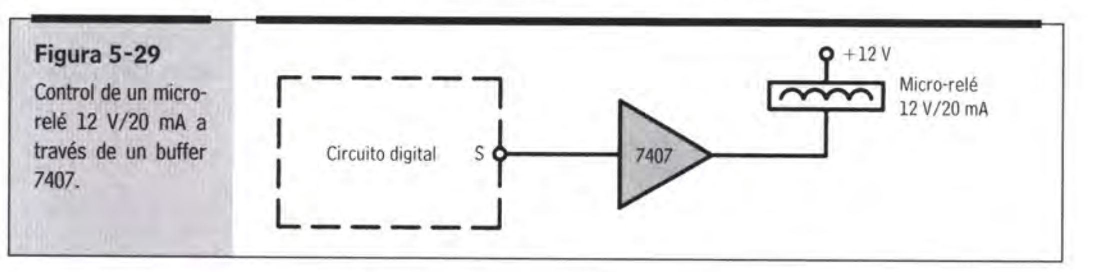


Figura 5-29
Control de un micro-relé 12 V/20 mA a través de un buffer 7407.

5.4.5. Disparador de Schmitt

Este dispositivo puede considerarse como una puerta peculiar. El estado alto o bajo de su salida es función de un determinado valor de la tensión de entrada. Así, la salida estará a nivel bajo si la entrada tiene una tensión menor que el valor especificado para el disparo, y será alta si es mayor que dicho valor.

El esquema básico de un circuito de disparo está constituido por dos inversores y un par de resistencias de interconexión, como se muestra en la figura 5-30.

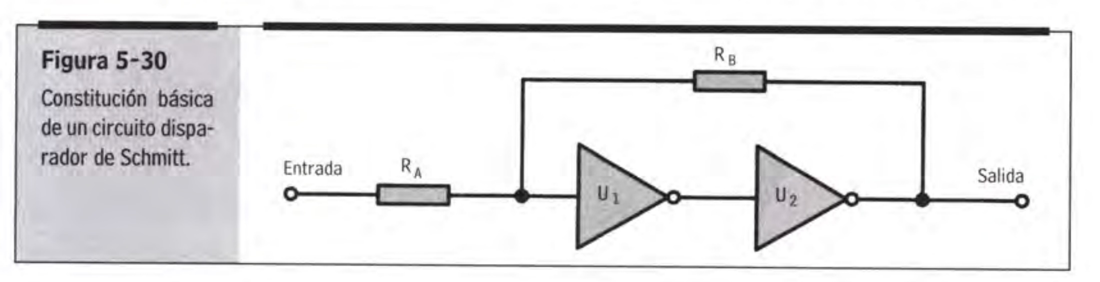


Figura 5-30
Constitución básica de un circuito disparador de Schmitt.

Como se observa en la figura 5-30, existe una realimentación de la salida a la entrada, que depende de los valores de R_A y R_B de forma que, cuando la tensión de entrada sube y comienza a bajar el nivel en la salida de U1 y a subir en la salida de U2, eleva la entrada a causa de la realimentación, disparándose y basculando el circuito. Según la relación entre R_A y R_B, queda determinado el valor de la tensión de referencia para la que se produce el basculamiento. En la figura 5-31 se muestran las formas de onda del circuito de disparo.

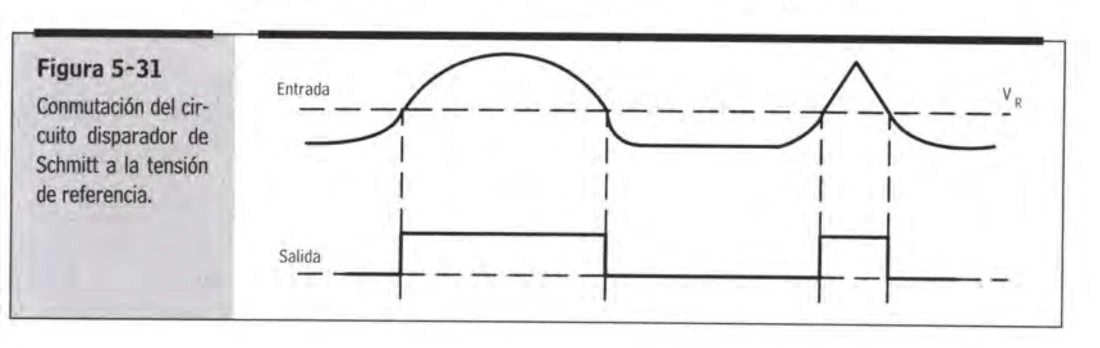
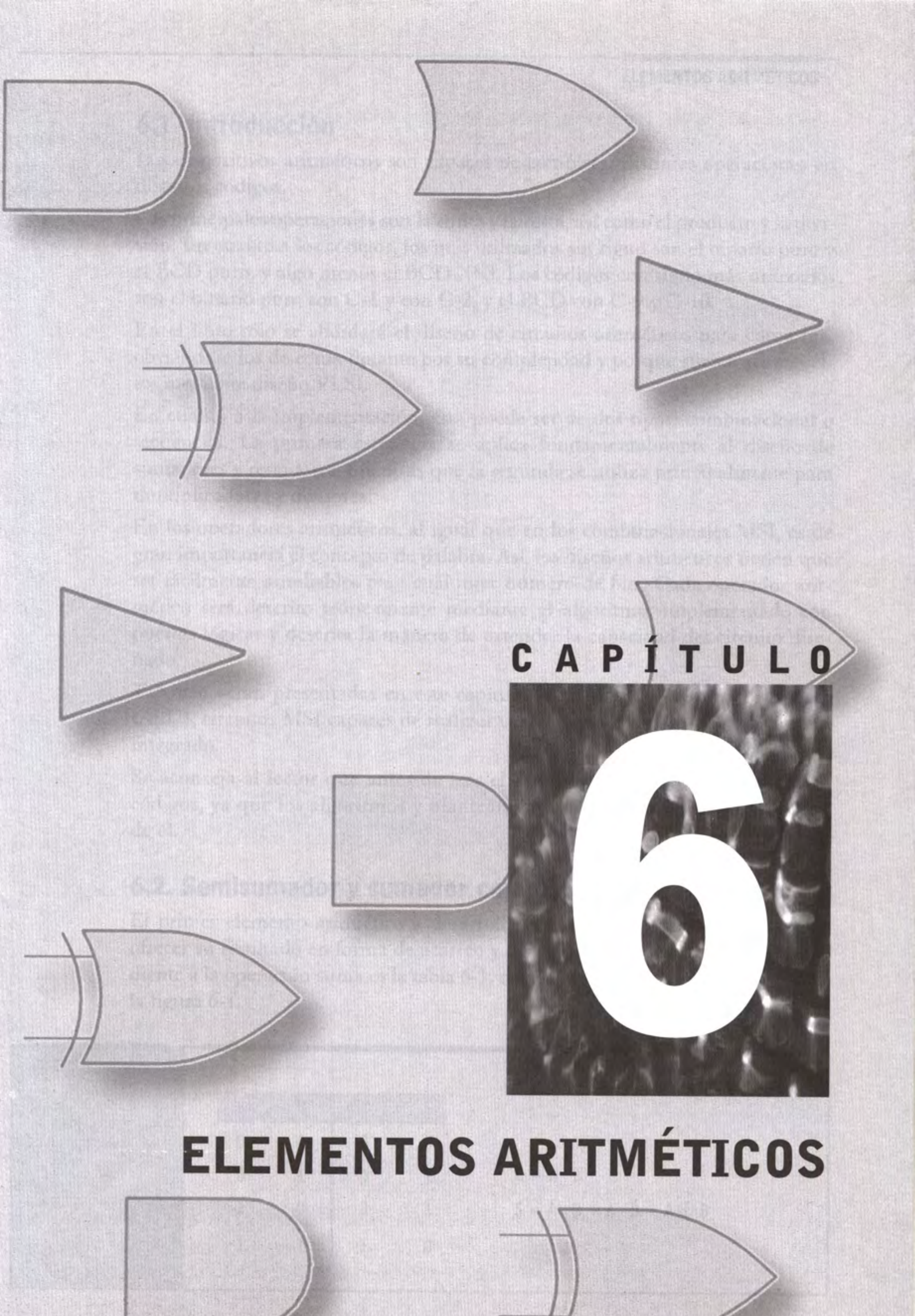


Figura 5-31
Conmutación del circuito disparador de Schmitt a la tensión de referencia.

Los disparadores de **Schmitt** se emplean como generadores de impulsos, detectores de nivel, conformadores de impulsos, adaptadores entre familias lógicas, etc. También se usan cuando cambian muy rápidamente los niveles, que no podrían aplicarse a las puertas TTL estándar porque podrían ocasionar salidas incorrectas o no definidas si los cambios en la entrada se produjeran antes de tomar la salida el estado correspondiente. Los disparadores de **Schmitt** se emplean para transformar señales analógicas en digitales.



CAPÍTULO

6

ELEMENTOS ARITMÉTICOS

6.1. Introducción

Los dispositivos aritméticos son capaces de completar distintas operaciones en distintos códigos.

Las principales operaciones son la suma y la resta, así como el producto y la división. En cuanto a los códigos, los más utilizados sin signo son el binario puro y el BCD puro, y algo menos el BCD XS3. Los códigos con signo más utilizados son el binario puro con C-1 y con C-2, y el BCD con C-9 y C-10.

En el libro sólo se abordará el diseño de circuitos aritméticos para coma fija, obviándose los de coma flotante por su complejidad y porque suelen ser resueltos mediante diseño VLSI.

En cuanto a la implementación, ésta puede ser de dos tipos: combinacional o secuencial. La primera estrategia se aplica fundamentalmente al diseño de sumadores y restadores, mientras que la segunda se utiliza principalmente para multiplicadores y divisores.

En los operadores aritméticos, al igual que en los combinacionales MSI, es de gran importancia el concepto de palabra. Así, los diseños aritméticos tienen que ser fácilmente ampliables para cualquier número de bits. Cada operador aritmético será descrito teóricamente mediante el algoritmo, implementado con puertas lógicas y descrita la manera de extender la capacidad del circuito diseñado.

También serán presentadas en este capítulo las Unidades Aritmético Lógicas (ALU), circuitos MSI capaces de realizar varias operaciones en un solo circuito integrado.

Se aconseja al lector que antes de leer el capítulo relea el correspondiente a códigos, ya que los algoritmos y planteamientos de cada operación proceden de él.

6.2. Semisumador y sumador completo

El primer elemento aritmético a desarrollar es aquel que suma dos bits para ofrecer su resultado en forma de acarreo y suma. La tabla de verdad correspondiente a la operación suma es la tabla 6-1, cuyo circuito lógico y bloque están en la figura 6-1.

Tabla 6-1

Tabla de verdad y ecuaciones del semisumador, HA.

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$C = A \cdot B$$

$$S = \bar{A} \cdot B + A \cdot \bar{B} = A \oplus B$$

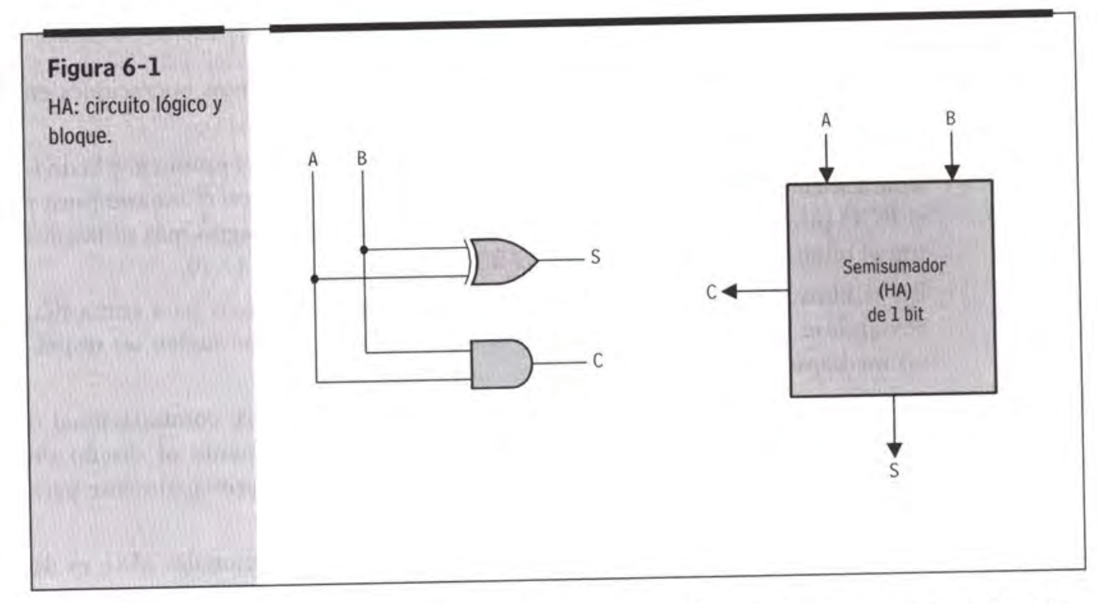


Figura 6-1
HA: circuito lógico y bloque.

A este dispositivo se le denomina semisumador -o HA, acrónimo de half adder- y tiene el inconveniente de que no permite la suma de varios bits, ya que no contempla como entrada el acarreo o llevada.

El sumador completo sí contempla el acarreo de entrada, lo que permite su extensión.

Un sumador completo -FA, full adder- se comporta como indica su tabla de verdad 6-2.

Tabla 6-2
T-V del sumador completo FA.

Ai	Bi	Ci-1	Si	Ci
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Su implementación puede tomar 3 caminos:

- Implementación como combinacional.
- Implementación utilizando la regla de la suma.
- Implementación basada en semisumadores.

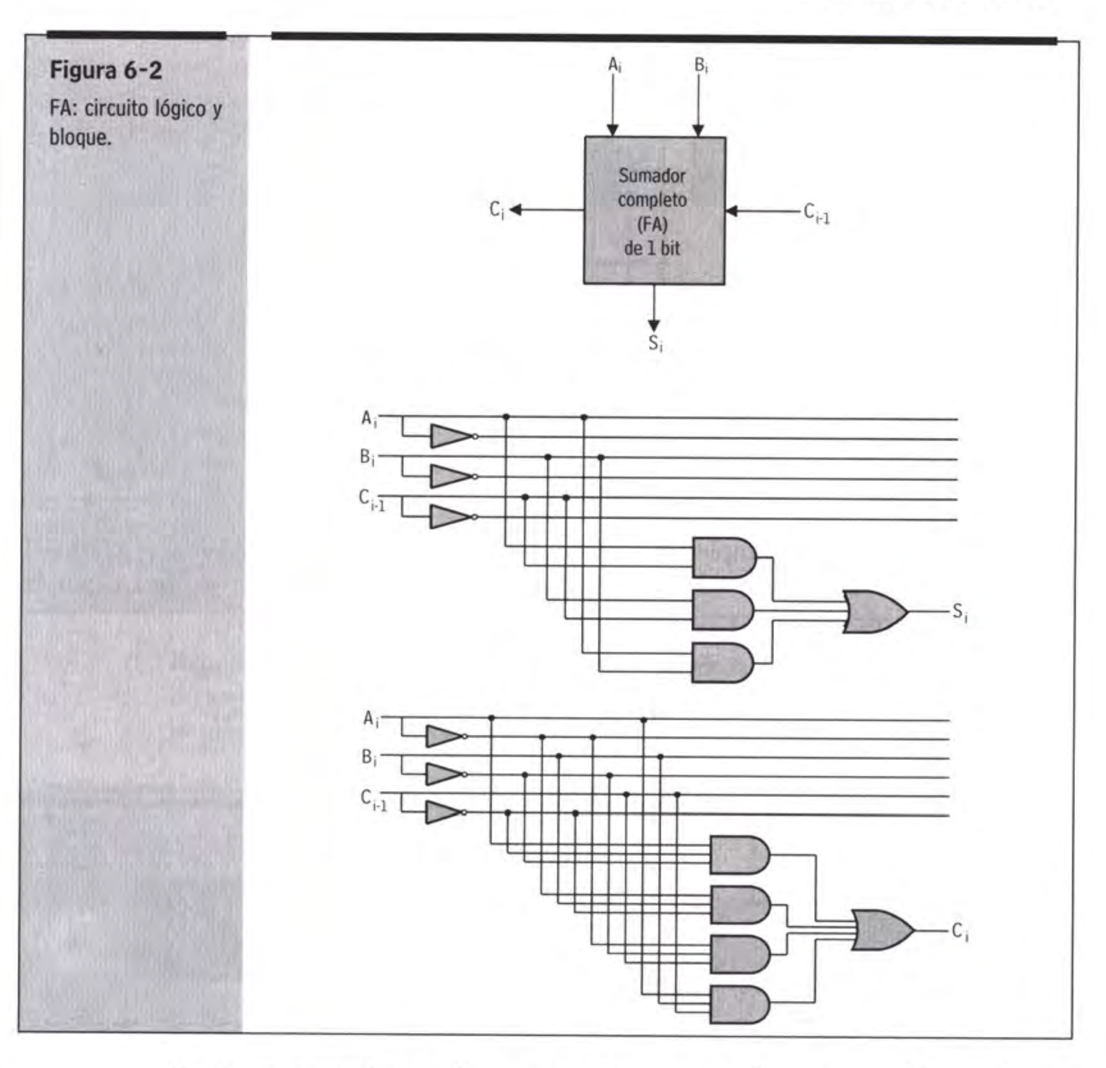


Figura 6-2

FA: circuito lógico y bloque.

En el primer camino resolvemos e implementamos los correspondientes diagramas de V-K de la tabla 6-2 obteniendo las ecuaciones (6.1) y el circuito de la figura 6-2.

$$S_i = A_i \cdot C_{i-1} + B_i \cdot C_{i-1} + A_i \cdot B_i \qquad (6.1)$$
$$C_i = A_i \cdot \overline{B_i} \cdot \overline{C_{i-1}} + \overline{A_i} \cdot B_i \cdot \overline{C_{i-1}} + \overline{A_i} \cdot \overline{B_i} \cdot C_{i-1} + A_i \cdot B_i \cdot C_{i-1}$$

En la segunda implementación hay que tener en cuenta que sumar tres bits es como sumar dos bits dos veces, y que el acarreo existe si al menos dos de los bits de entrada son 1. Si reescribimos lo anterior en forma booleana tenemos la ecuación 6.2 y el circuito de la figura 6-3.

$$S_i = (A_i \oplus B_i) \oplus C_{i-1} \qquad (6.2)$$
$$C_i = A_i \cdot B_i + (A_i \oplus B_i) \cdot C_{i-1}$$

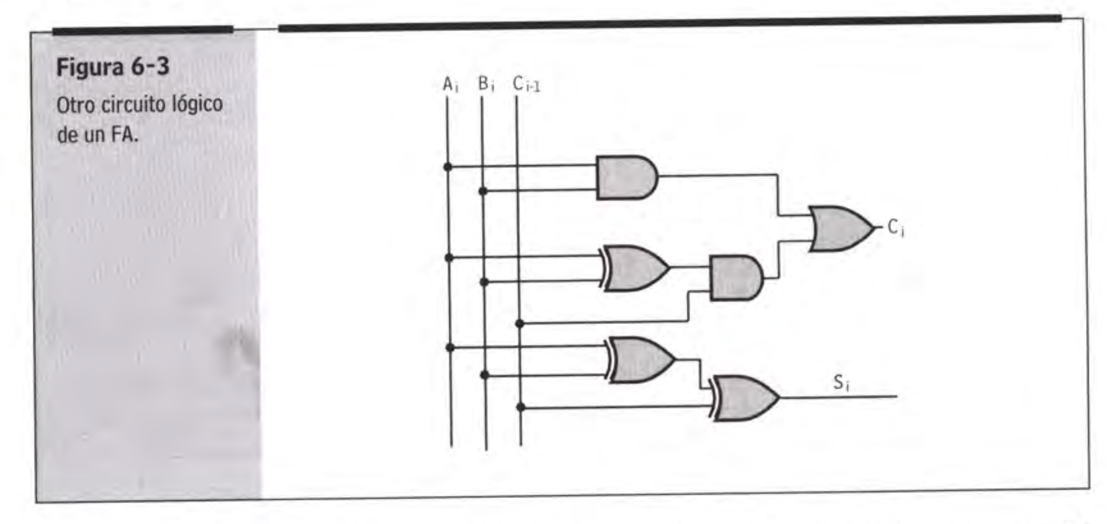


Figura 6-3
Otro circuito lógico de un FA.

En la tercera simplemente reordenaremos las anteriores ecuaciones respecto del semisumador, resultando la implementación de la figura 6-4, que, como se puede observar, respeta las ecuaciones 6.1 y 6.2.

$$S'_i = A_i \oplus B_i \quad y \quad C'_i = A_i \cdot B_i$$

$$S''_i = S'_i \oplus C_{i-1} \quad y \quad C''_i = S'_i \cdot C_{i-1} = (A_i \oplus B_i) \cdot C_{i-1}$$

$$S = S''_i \quad C_i = C'_i + C_{i-1}$$

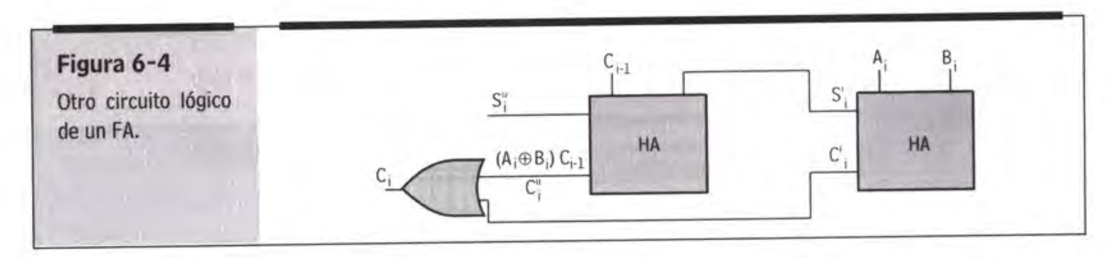


Figura 6-4
Otro circuito lógico de un FA.

Las tres implementaciones son válidas, aunque con distinto coste en tiempo y puertas y distinto comportamiento frente a los riesgos lógicos. Su planteamiento nos permite observar cómo de distintos enfoques se obtienen soluciones correctas; situación ésta, que aunque es puramente teórica, es muy típica en los circuitos aritméticos.

Toda vez que tenemos un sumador completo de 1 bit podemos abordar un sumador de cualquier número de bits.

6.3. Sumador en paralelo con acarreo en serie

Esta implementación concuerda con nuestra forma de sumar: primero sumo dos dígitos, luego los dos siguientes más el acarreo anterior, etc. Este mecanismo es fácilmente implementable con sumadores completos; la figura 6-5 muestra el esquema y el bloque de un sumador de 4 bits.

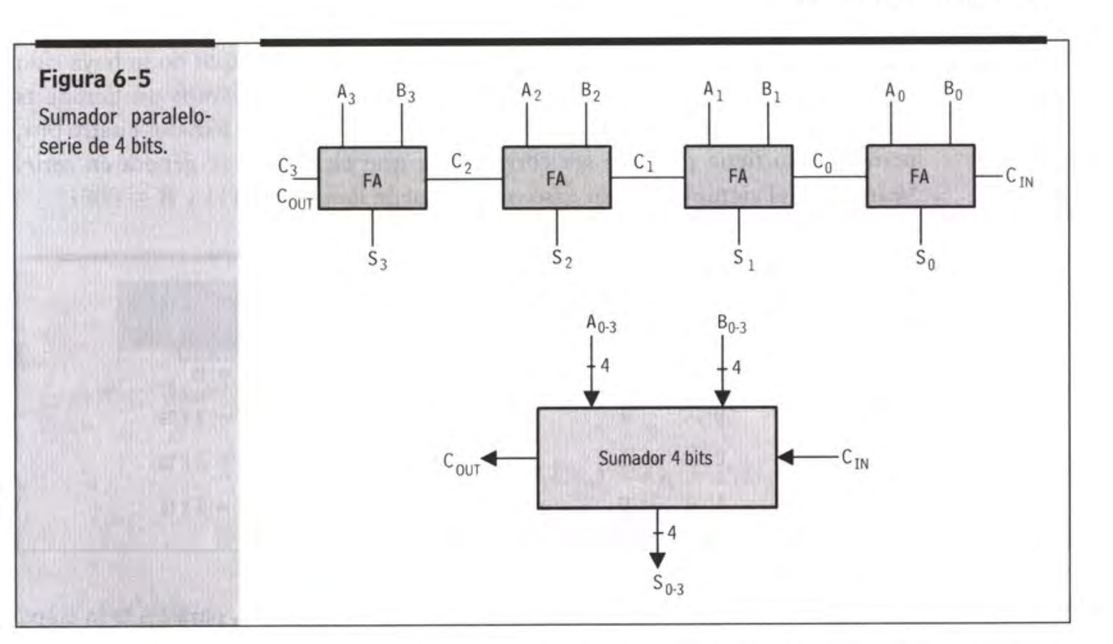


Figura 6-5
Sumador paralelo-serie de 4 bits.

Repitiendo el planteamiento anterior, un sumador de 8 bits se obtiene conectando en serie dos sumadores de 4 bits (figura 6-6), y del mismo modo se procederá para mayor número de bits.

Figura 6-6
Sumador paralelo-serie de 8 bits.

Al circuito de la figura 6-6, cualquiera que sea el número de bits, se le denomina sumador paralelo con acarreo serie porque las entradas se presentan simultáneamente, en paralelo, mientras que el acarreo se produce en serie, etapa a

etapa. Por ejemplo, el acarreo C_2 no puede generarse hasta que no lo haya sido el C_1, y así sucesivamente. Lo anterior supone que si asociamos un tiempo ts para sumar 1 bit, pasado este tiempo tendremos un resultado para los cuatro bits, pero éste no tiene por qué ser correcto, ya que el acarreo se genera en serie. Veamos en el ejemplo el peor caso que se puede dar: A = 1111 y B = 0001.

Tabla 6-3
Evolución temporal de la suma en el peor caso posible.

	1	1	1	1	
+	0	0	0	1	
0	1	1	1	0	t = ts
0	1	1	0	0	t = 2 x ts
0	1	0	0	0	t = 3 x ts
1	0	0	0	0	t = 4 x ts

El resultado no es correcto hasta pasados 4 x ts. Por ejemplo, para t ≤ ts la etapa 2 suma $A_1 = 1$, $B_1 = 0$ y $C_0 = 0$, pues todavía no se ha generado correctamente el C_0, resultando que la suma es 01110. Así pues, la suma es correcta sólo si t > 4 x ts.

Por tanto, para obtener el resultado correcto de sumar n bits hay que esperar a que transcurran los n x ts segundos correspondientes al caso más desfavorable (otros casos necesitarían menos tiempo, pero no son generalizables).

$$\text{Tiempo de suma de n bits} = n \times ts \qquad (6.3)$$

Si el número de bits es elevado -a partir de 16 bits- este tiempo de retardo no es despreciable, convirtiéndose en muchos casos en inaceptable. Para resolver esta situación se plantean circuitos más rápidos, aunque lo sean a costa de una mayor complejidad y coste circuital.

6.4. Sumador paralelo con acarreo anticipado

En este circuito los acarreos se generan en paralelo al igual que las entradas, por tanto se reduce el tiempo total de suma.

Para plantear el circuito comencemos por recordar que en un sumador completo la ecuación correspondiente al acarreo es $C_i = A_i \cdot B_i + (A_i \oplus B_i) \cdot C_{i-1}$. Al observar esta ecuación podemos decir que $A_i \cdot B_i$ es el producto que genera el acarreo -y así lo denominamos G_i-, mientras que de $(A_i \oplus B_i)$ podemos decir que propaga el acarreo de la anterior etapa -y lo denominamos P_i-, ya que si $P_1 = 1$ se propaga el valor de C_{i-1}. Reescribamos en 6.4 la ecuación correspondiente al acarreo.

$$\begin{aligned} G_i &= A_i \cdot B_i \\ P_i &= A_i \oplus B_i \\ C_i &= G_i + P_i \cdot C_{i-1} \end{aligned} \qquad (6.4)$$

Teniendo en cuenta lo anterior planteemos de nuevo las ecuaciones correspondientes para un sumador de 4 bits, donde podemos representar los acarreos de salida de las cuatro etapas como:

$$C_0 = G_0 + P_0 \cdot C_{IN}$$
$$C_1 = G_1 + P_1 \cdot C_0$$
$$C_2 = G_2 + P_2 \cdot C_1$$
$$C_3 = G_3 + P_3 \cdot C_2$$

Ahora bien, estas ecuaciones están serializadas, pues C_i se genera a partir de C_{i-1}. Para paralelizarlas simplemente reescribimos las ecuaciones como indica 6.5.

$$C_0 = G_0 + P_0C_{IN}$$
$$C_1 = G_1 + P_1\,(G_0 + P_0C_{IN}) = G_1 + P_1G_0 + P_1P_0C_{IN} \qquad (6.5)$$
$$C_2 = G_2 + P_2\,(G_1 + P_1G_0 + P_1P_0C_{IN}) =$$
$$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{IN}$$
$$C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{IN}$$

La implementación del sumador correspondiente a las ecuaciones 6.5 es el de la figura 6-7. A primera vista puede parecer complejo de obtener, pero fijándose no es así. En la figura 6-7 vemos dos líneas no contempladas en las ecuaciones, G y P, que serán explicadas más adelante.

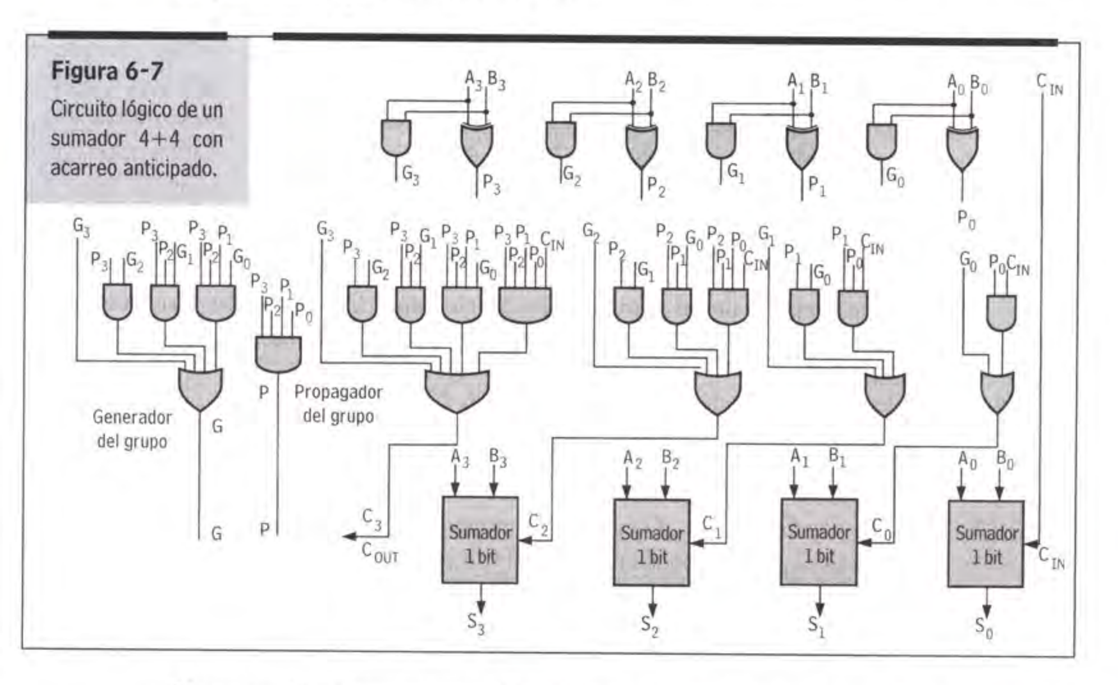


Figura 6-7
Circuito lógico de un sumador 4+4 con acarreo anticipado.

Una cuestión importante a dilucidar es: ¿cuánto tiempo necesita un sumador con acarreo anticipado para completar una suma? Para determinarlo, fijémonos en tres aspectos:

- la obtención en paralelo de Pi y Gi,
- la obtención en paralelo de Ci,
- la suma en cada etapa toda vez que se ha obtenido Ci.

La obtención de P_i y G_i consume 2 x tp (tp = tiempo puerta), a esto hay que sumar otros 2 x tp por la obtención de C_i; así cualquier acarreo está disponible en 4 x tp. Para obtener S_i hay que sumar 2 x tp -toda vez que consideramos que una XOR viene a tardar el doble que una puerta normal-. La suma por tanto se obtiene de 6 x tp segundos, sea cual sea el número de bits; de hecho en la expresión desaparece el término n. La tabla 6-4 compara ambos sumadores.

Tabla 6-4
Comparación entre sumador serie y paralelo.

Sumador con acarreo serie	Sumador con acarreo paralelo
Tsuma = n x 2tp	Tsuma = 6 x tp
Tacarreo = (n + 1) x 2tp	Tacarreo = 4 x tp = TGP de grupo

Volvamos a la figura 6-7, donde las líneas G y P de grupo permiten extender la capacidad de los sumadores como veremos más adelante. La línea G se pone a 1 cuando dicha etapa o grupo ha generado un acarreo, y P se pone a 1 cuando dicha etapa ha permitido la propagación del acarreo de entrada en la etapa. La obtención de estas líneas retarda el proceso hasta 5 x tp.

Como ya hemos dicho, en los tiempos de retardo de este circuito ha desaparecido el término n, a cambio el circuito se ha complicado hasta el punto de que es muy extraño que un sumador con acarreo anticipado supere los 8 bits. Cuando haya que implementar sumadores de mayor número de bits y no se pueda aceptar el retardo del sumador con acarreo serie, se optará por implementaciones híbridas basadas en un nuevo circuito: el LAC (LookAhead Carry).

6.5. Técnicas híbridas en sumadores

La complejidad de un sumador con acarreo anticipado para un gran número de bits hace que planteemos soluciones que no son ni puramente paralelas ni puramente serie.

6.5.1. Circuito LAC

Un circuito LAC, según las ecuaciones 6.5, tiene como entradas nueve líneas: P_0, P_1, P_2, P_3, G_0, G_1, G_2, G_3 y C_{IN}, y genera como salida los acarreos: C_0, C_1 y C_2, y a veces C_3, y dos líneas auxiliares P y G de grupo. La figura 6-8 es el bloque de un LAC.

$$
\begin{aligned}
C_0 &= G_0 + P_0C_{IN} \\
C_1 &= G_1 + P_1G_0 + P_1P_0C_{IN} \qquad (6.5) \\
C_2 &= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{IN} \\
C_3 &= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{IN}
\end{aligned}
$$

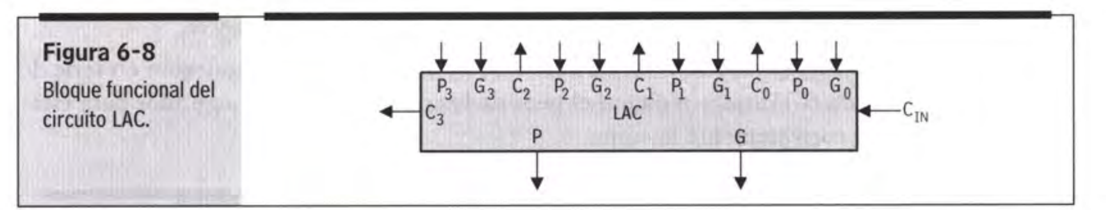


Figura 6-8
Bloque funcional del circuito LAC.

Utilizando el LAC anterior y el sumador con generación/propagación de la figura 6-9 podemos implementar un sumador 4 + 4 con acarreo anticipado como el de la figura 6-10.

Figura 6-9
Sumador 4 + 4 con acarreo anticipado por LAC.

Las líneas de acarreo de salida del primer nivel de la figura 6-9 no se usan, excepto C_3 que genera el C_{OUT} del circuito total. La razón estriba en que el coste de C_3 es muy elevado en el LAC, es decir, aunque aparezca C_3 en el LAC de la figura 6-7, éste no suele estar disponible en los circuitos reales.

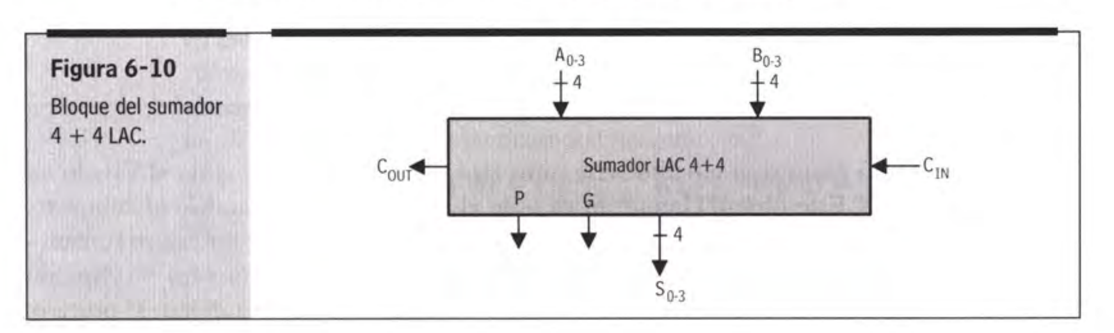


Figura 6-10
Bloque del sumador 4 + 4 LAC.

6.5.2. Sumadores con acarreo anticipado conectados en serie

Por ejemplo, pensemos en un sumador de 16 + 16 bits. La conexión en serie de la figura 6-11 necesitaría, en el peor de los casos, de 32 x tp segundos para establecer correctamente la suma.

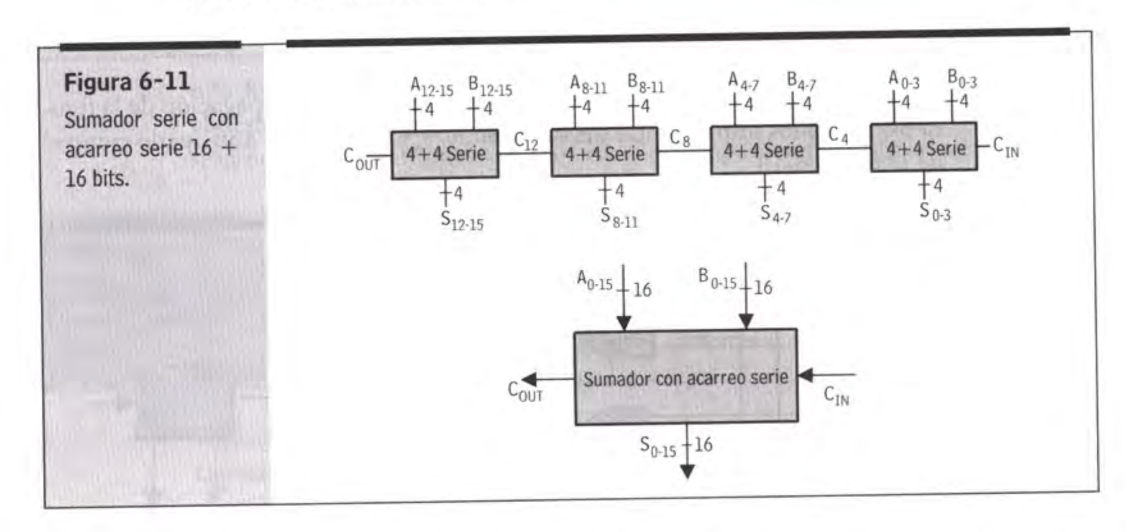


Figura 6-11 Sumador serie con acarreo serie 16 + 16 bits.

Si sustituyéramos los sumadores con acarreo serie por sumadores LAC la conexión final sería la de la figura 6-12. Es decir, dentro de las etapas en paralelo y entre etapas en serie.

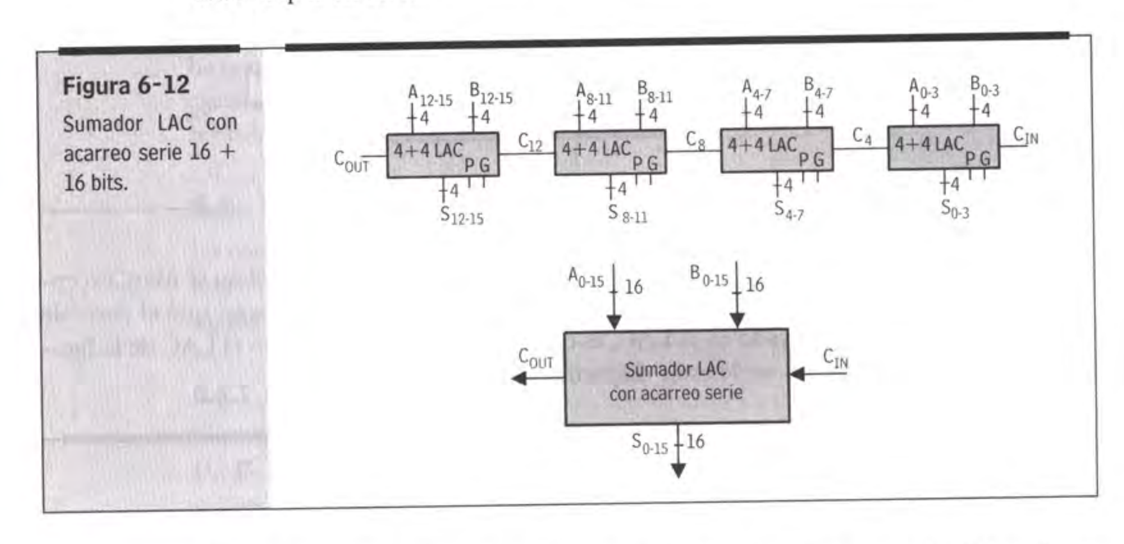


Figura 6-12 Sumador LAC con acarreo serie 16 + 16 bits.

Ahora bien, si el circuito serie puro tarda 32 x tp, ¿cuánto tarda el basado en LAC? Este último circuito ni es serie al completo, ni es paralelo al completo. Veamos, cada sumador LAC consume 6 x tp segundos, y por ser cuatro sumadores en serie podríamos pensar que el retardo final sería 4 x (6 x tp) = 24tp. Sin embargo no es así, ya que se producen solapamientos que aceleran el proceso:

durante los primeros 2 x tp se generan simultáneamente todos los G_i y P_i, pasados otros 2 x tp -el acarreo tiene dos niveles- se genera C_8, igualmente para C_{12} y C_{16}; así, este último se obtendrá al de 10 x tp. Resumiendo, el acarreo tarda:

$$Tacarreo = 2 \times tp + n \times 2tp = 10 \times tp, \text{ para } n = 4 \quad (6.6)$$

Además, simultáneamente a la generación del acarreo se va obteniendo la suma, excepto el último bit que necesita 2 x tp segundos adicionales, como ya hemos visto.

$$Tsuma = 2 \times tp + n \times 2 \times tp + 2 \times tp = (n + 2)\ ts =$$
$$12 \times tp = 6 \times ts, \text{ para } n = 4, \text{ donde } ts = 2 \times tp$$

$$Tsuma = (n + 2)\ ts = (n + 2) \times (2 \times tp), \quad (6.7)$$

donde n es el nº de bits

Si comparamos este tiempo con el del sumador serie vemos que la reducción es de un 62,5% para 16 bits. En general y con sumadores de 4 bits:

$$\frac{Tsumaserie}{TsumaLACserie} = \frac{n \times ts}{(m + 2) \times ts} = \frac{n}{\frac{n}{4}+2} \approx 4, \frac{n}{4} >> 2$$

donde n = número bits y (6.8)
m = número sumadores de 4 bits

La anterior expresión cuantifica que el sumador LAC con conexión serie es del orden de 4 veces más rápido que el serie puro, y no tiene sentido hablar del LAC puro, ya que es difícilmente implementable.

6.5.3. Sumadores con acarreo anticipado en un LAC

En este caso y como muestra la figura 6-13 el acarreo de la suma dentro del sumador se propaga en paralelo, y el acarreo para la suma entre etapas es generado en paralelo. Es decir, dentro de las etapas en paralelo y entre las etapas en paralelo, pero no en paralelo total.

Analicemos el circuito temporalmente, fase por fase. Primeramente, los cuatro LAC's del nivel superior obtienen en paralelo en 4 x tp segundos -tgp(4 + 4)- las líneas P y G. En la segunda fase, el LAC del nivel inferior obtiene de las líneas P y G en 2 x tp -$t_{acarreoLAC}$- y los acarreos C_0, C_1 y C_2, que en este caso son C_4, C_8 y C_{12}. Así, pasados 6 x tp segundos los acarreos vuelven a los LAC's del nivel superior para completar la suma. Estos cuatro LAC's del nivel superior suman en paralelo entre sí, cada uno de ellos tarda 2 x tp para los acarreos internos y las sumas de los tres primeros bits, necesitando 2 x tp segundos adicionales para la suma del cuarto bit, en total $t_{suma}(4 + 4) = 4 \times tp$. El tiempo total consumido es de 10 x tp segundos. Generalizando obtenemos 6.9.

$$Tsuma = t_{gp}\ (4 + 4) + t_{acarreoLAC} + t_{suma}\ (4 + 4) =$$
$$= 4 \times tp + 2 \times tp + 4 \times tp = 10 \times tp \text{ segundos}$$

$$Tsuma = 10 \times tp = 5 \times ts \quad (6.9)$$

sólo si el número de sumadores del primer nivel es menor o igual que cuatro.

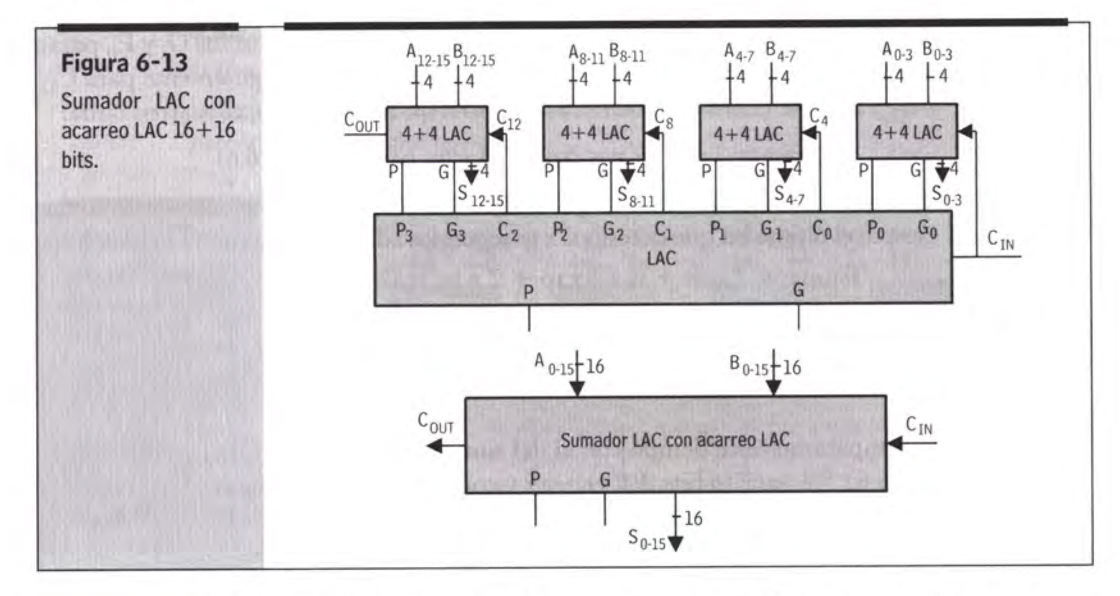


Figura 6-13
Sumador LAC con acarreo LAC 16+16 bits.

Al obtener la anterior expresión puede parecer que el circuito es totalmente paralelo, pero no es así, ya que la unidad de LAC tiene un límite físico en cuatro grupos P y G. Si el circuito superara esta disponibilidad habría que plantear una extensión del anterior circuito, sin más que aplicar un criterio recursivo.

Por ejemplo, el sumador de 64 bits de la figura 6-14 es una agrupación de cuatro sumadores de 16 + 16 como el diseñado en la figura 6-13.

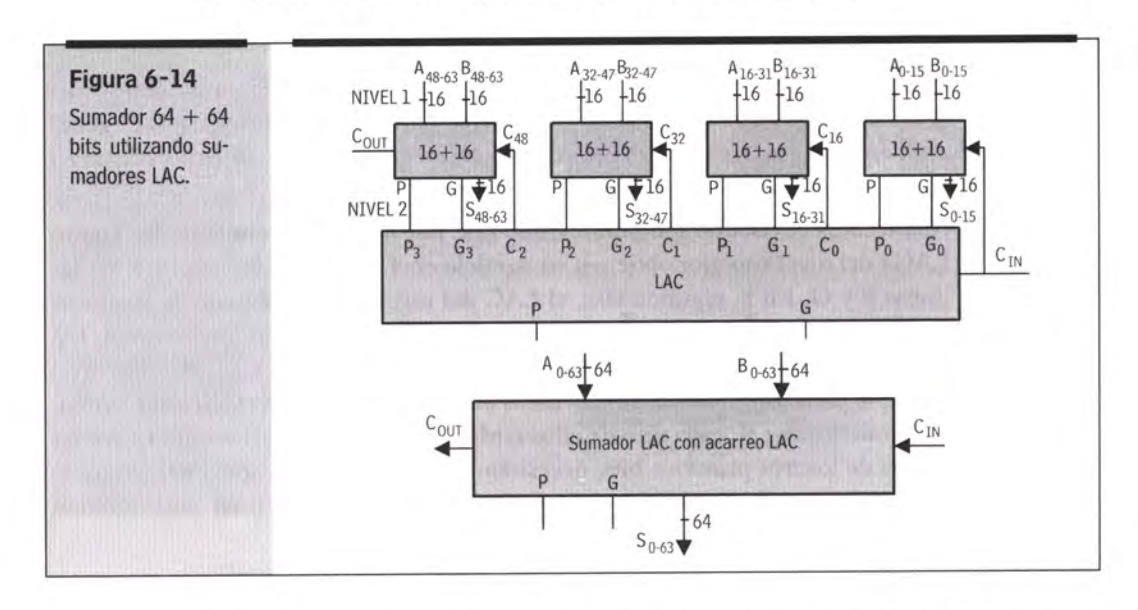


Figura 6-14
Sumador 64 + 64 bits utilizando sumadores LAC.

En este caso el tiempo consumido sería 4 x tp para las líneas P y G de los sumadores LAC y otros 4 x tp para las líneas P y G de grupo del primer nivel (ambas incluidas en el 16 + 16). Pasados 8 x tp el LAC del segundo nivel recibe las líneas

P y G, y en 2 x tp ofrece los acarreos a las etapas 4 + 4 que a su vez consumirán 2 x tp en generar la suma y otros 2 x tp en establecer la suma del último bit.

$$Tsuma = t_{gp} + t_{acarreo} + t_{suma}(4 + 4) = 4 \text{ x tp} + 4 \text{ x tp} + 2 \text{ x tp} + 4 \text{ x tp} = 14 \text{ x tp}$$

El valor 14 x tp obtenido contrasta fuertemente con los 130 x tp segundos necesarios para establecer la suma con acarreo serie. La expresión generalizada para esta técnica de suma recursiva es:

$$Tsuma = \left(\log_2 \frac{n}{16}\right) 4 \text{ x tp} + 2 \text{ x tp} + 4 \text{ x tp} \qquad (6.10)$$

con n=nº de bits

Obsérvese cómo la relación que antes era lineal respecto de n, ahora es logarítmica.

6.5.4. Comparación entre distintas implementaciones de sumadores

Como hemos visto, existen distintas implementaciones para un sumador con entrada en paralelo: todas ellas obtienen la suma de n bits, pero de distinta forma. Principalmente hay cuatro técnicas:

- Sumador paralelo con acarreo serie.
- Sumador paralelo con acarreo anticipado. Sumador LAC puro.
- Conexión en serie de sumadores de acarreo anticipado. Sumador híbrido de LAC's en serie.
- Conexión por niveles de sumadores LAC. Sumador híbrido de LAC's por niveles.

Para establecer la comparación nos fijaremos en determinados aspectos:

- Constructibilidad. Indica si un sumador es construible, cualquiera que sea el número de bits.
- Extensibilidad. Indica el grado de facilidad para extender la capacidad de un sumador.
- Rapidez. Indica cuánto se tarda en obtener la suma de n bits.
- Dependencia. Indica de qué tipo es la dependencia temporal del sumador.
- Sencillez. Indica en qué medida son sencillos los sumadores.

La tabla 6-5 compara las cuatro técnicas para sumar en binario puro.

Tabla 6-5

Tabla comparativa entre distintas técnicas de suma.

	Construct.	Extensib.	Rapidez	Dependencia	Sencillez
Serie	Sí	Muy fácil 1º	$(n + 1) \cdot 2tp$	Lineal depende de n	1º
LAC	No	Difícil 4º	6tp	Independiente	4º
LAC en serie	Sí	Muy fácil 2º	$\left(\frac{n}{4} + 2\right) \cdot 2tp$	Lineal depende de $\frac{n}{4}$	2º
LAC en LAC	Sí	Fácil 3º	$\left(\log_2 \frac{n}{16}\right) \cdot 4tp + 6tp$	Logarítmica depende $\left(\log_2 \frac{n}{16}\right)$	3º

6.6. Restadores en binario con signo

Si para diseñar un sumador planteamos el circuito correspondiente a un sumador de 1 bit, podemos repetir el mismo procedimiento para un restador. En un restador de 1 bit se restan dos bits y el correspondiente préstamo para obtener la resta y el préstamo. La tabla de verdad, las ecuaciones booleanas y el circuito correspondiente son la tabla 6-6, las ecuaciones 6.11 y la figura 6-15.

Tabla 6-6
T-V de un restador de un bit.

A	B	PIN	POUT	R
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Resolviendo los diagramas de V-K obtenemos las ecuaciones correspondientes a un restador de 1 bit.

$$P_{OUT} = \overline{A} \cdot P_{IN} + B \cdot P_{IN} + \overline{A} \cdot B \qquad (6.11)$$

$$R = A \cdot \overline{B} \cdot \overline{P_{IN}} + \overline{A} \cdot B \cdot \overline{P_{IN}} + \overline{A} \cdot \overline{B} \cdot P_{IN} + A \cdot B \cdot P_{IN} =$$

$$= \overline{P_{IN}} \, (A \cdot \overline{B} + \overline{A} \cdot B) + P_{IN} \cdot (\overline{A} \cdot \overline{B} + A \cdot B) = A \oplus B \oplus P_{IN}$$

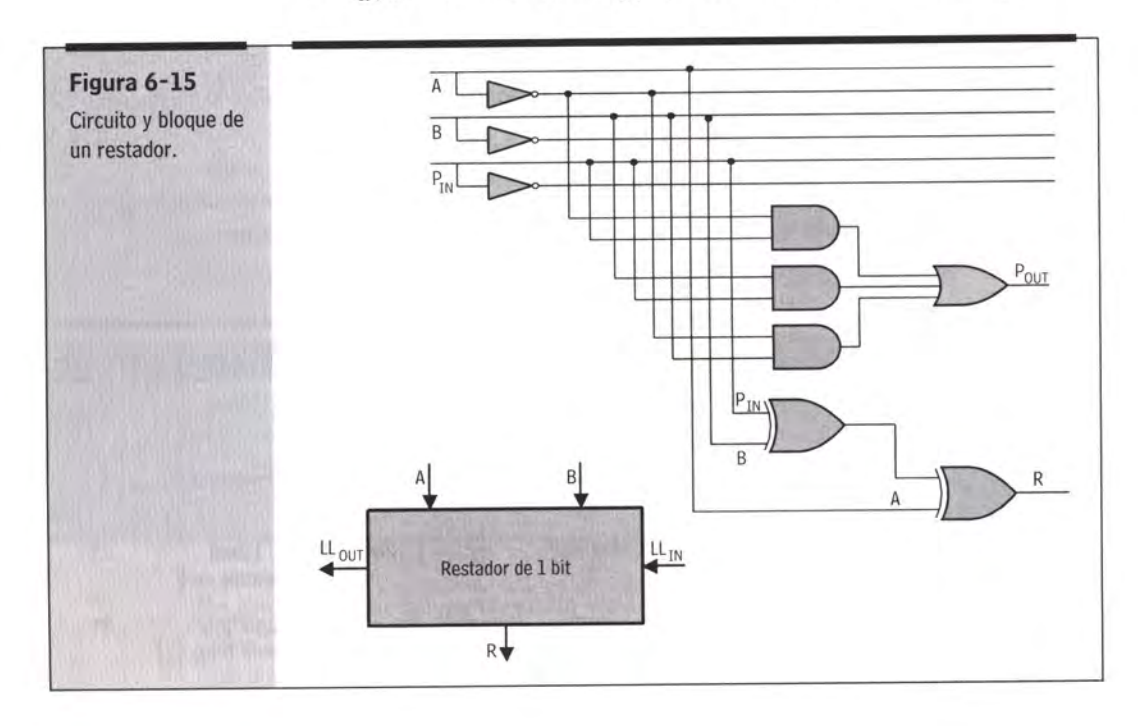


Figura 6-15
Circuito y bloque de un restador.

Para obtener un restador de 4 bits sólo queda unir en serie cuatro restadores de 1 bit, como muestra la figura 6-16.

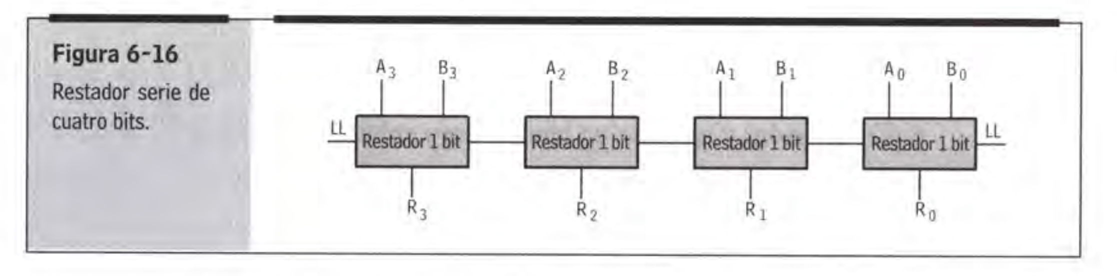


Figura 6-16
Restador serie de cuatro bits.

Como podemos observar, podríamos seguir una estrategia idéntica a la utilizada en la suma. Sin embargo, en vez de seguir este camino paralelo, resulta más cómodo reescribir la resta como una suma con signo en 6.12.

$$A - B = A + (-B) \qquad (6.12)$$

Según la anterior expresión la resta se convierte en el siguiente algoritmo:

- Tomar A y B.
- Cambiar de signo el sustraendo B' = (-B).
- Sumar A y B'.

Este planteamiento es muy sencillo en su raíz, pero se complica en su desarrollo ya que hay muchas formas de representar el signo en el sistema binario. En los siguientes apartados vamos a plantear la suma y resta de números con signo codificados en:

- Magnitud y bit de signo.
- Binario puro y complemento a 1.
- Binario puro y complemento a 2.
- Binario puro con exceso a la potencia.
- Binario puro con exceso a la potencia menos 1.

Así pues, en los siguientes apartados el diseño correspondiente a la resta se reescribe como el diseño de una suma donde los operandos tienen signo. Además aprovecharemos para plantear aquellos circuitos versátiles, capaces de realizar tanto restas como sumas en el código determinado, dando lugar a los Sumadores/Restadores. También abordaremos en estos códigos la detección del desborde en la operación, es decir, determinar en qué situación el resultado obtenido no puede escribirse con el número de bits original, y por tanto el resultado obtenido es erróneo y no utilizable. En los sumadores anteriormente vistos el desborde coincidía con el valor del C_{OUT}, pero en los siguientes circuitos su determinación no es tan espontánea.

Para todos los diseños siguientes seguiremos un esquema parecido:

- Manejo del signo en ese código.
- Determinación del algoritmo correspondiente.
- Obtención del desborde u overflow.

- Implementación del circuito restador para 4 bits.
- Tiempo consumido en la operación.
- Estudio de la extensibilidad del circuito.
- Ejemplo aclaratorio si fuera necesario.
- Implementación del circuito sumador/restador para 4 bits.

6.6.1. Sumador y restador en binario puro con C-1

En este caso un número negativo se escribe como el complemento a 1 del correspondiente positivo, así el algoritmo de la resta es:

- Tomar A (minuendo) y B (sustraendo).
- Complementar a 1 (negar) el sustraendo B'= C-1(B).
- Obtener la resta como R = A + B', recirculando el acarreo.

Estudio del desborde

La suma A+B' puede desbordarse sólo si A y B' son positivos o negativos.

Concretemos el estudio para cuatro bits, donde el rango es [-7,+7]:

- Si A y B' son positivos, en ambos su bit de signo (A_3 y B_3) es cero, resultando que la suma ha de ser positiva, y su bit de signo (S_3) cero. Por tanto, habrá overflow si al sumar $S_3 = 0 + 0 + C_2 = 1$, lo que sólo puede ser cierto si $C_2 = 1$. Reescribiendo lo anterior booleanamente:

  $$\text{Des. positivo} = \overline{A_3} \cdot \overline{B_3} \cdot C_2 = \overline{C_3} \cdot C_2$$

- Si A y B' son negativos en ambos su bit de signo (A_3 y B_3) es uno, resultando que la suma ha de ser negativa, y su bit de signo (S_3) uno. Por tanto, habrá overflow si al sumar $S_3 = 1 + 1 + C_2 = 0$, esto sólo puede ser cierto si $C_2 = 0$. Reescribiendo lo anterior booleanamente:

  $$\text{Des. negativo} = A_3 \cdot B_3 \cdot \overline{C_2} = C_3 \cdot \overline{C_2}$$

- Si A es positivo y B' es negativo, o viceversa, no puede darse en ningún caso desborde.
- Reuniendo el desbordamiento negativo y positivo resulta 6.13.

  $$\text{Desborde} = \overline{C_3}\, C_2 + C_3\, \overline{C_2} = C_3 \oplus C_2$$

  $$DE = C_{n-1} \oplus C_{n-2}, \qquad (6.13)$$

  donde n = nº de bits.

La implementación del restador en BP-C_1 aparece en la figura 6-17.

En la figura 6-17 observamos que el circuito complementador a 1 simplemente consta de 4 inversores, y que es necesario recircular el acarreo final para que el resultado sea correcto. En el bloque final podemos observar cómo éste no tiene C_{IN} (en nuestro caso P_{IN}), lo que quiere decir que no es directamente extensible al modo del sumador con acarreo serie, aunque por supuesto sería fácil implementar un restador de 8 bits, como el de la figura 6-18.

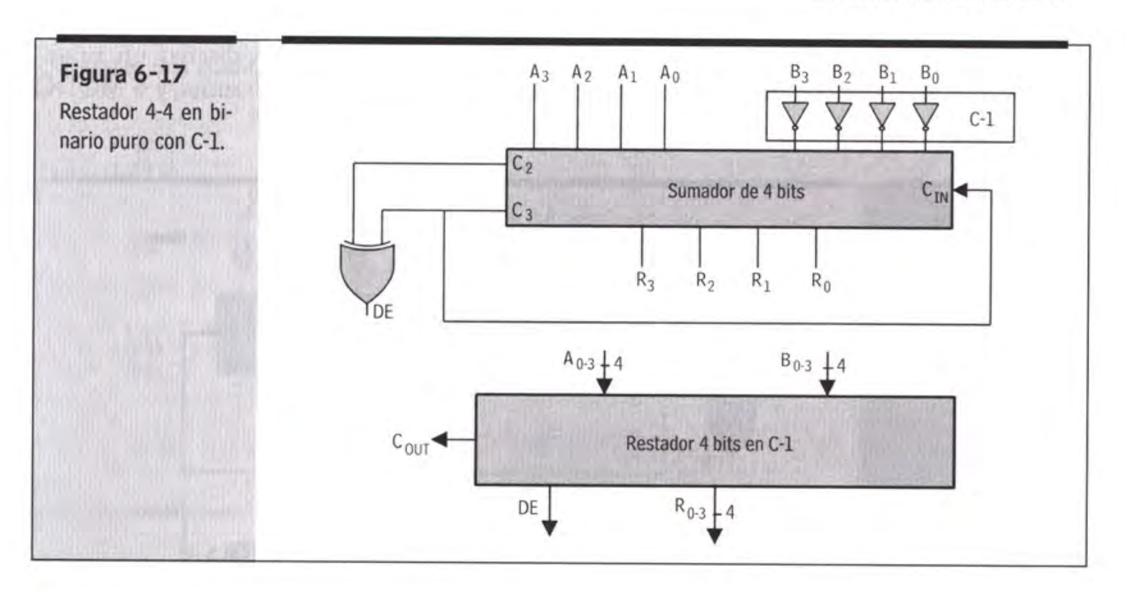


Figura 6-17
Restador 4-4 en binario puro con C-1.

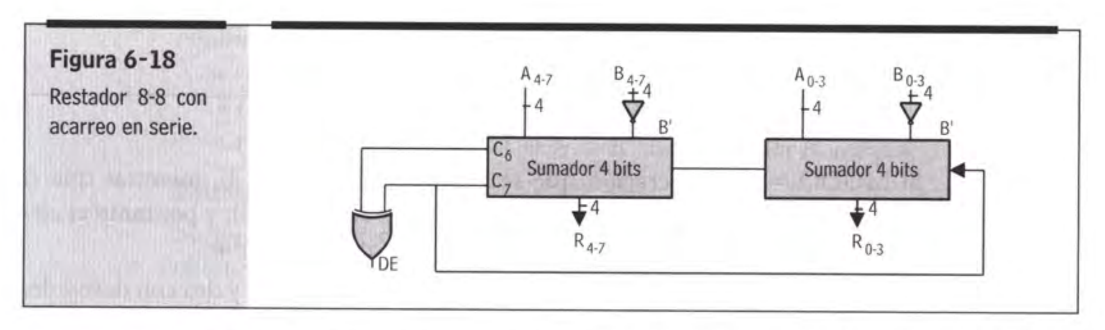


Figura 6-18
Restador 8-8 con acarreo en serie.

Respecto del tiempo consumido en la resta no es, como en la suma, (n+1) x ts; sino que el hecho de recircular el acarreo hace que el retardo asociado a la resta sea 2x (n + 1) x ts, es decir, el doble de una suma en binario puro.

6.6.1.1. Sumador/Restador

Como hemos visto no tiene sentido distinguir entre las operaciones suma y resta, sólo se diferencian en que en la resta el operando B debe ser complementado a 1, y en la suma no. La puerta XOR se adapta a esta necesidad, pues para $R = A \oplus AUX$ resulta que si $AUX = 0$ entonces $R = A$, y que si $AUX = 1$ entonces $R = \overline{A}$.

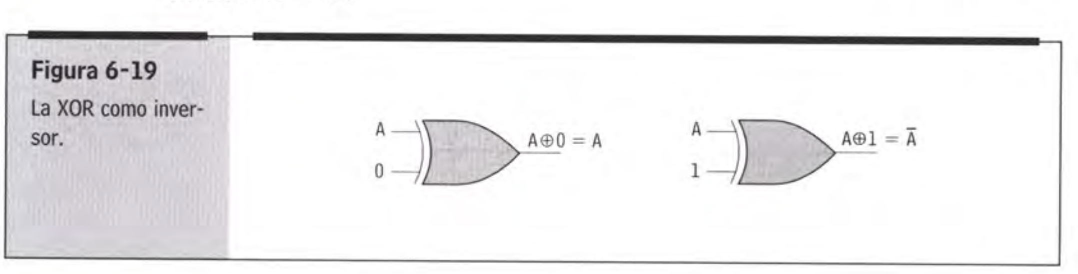


Figura 6-19
La XOR como inversor.

Así, utilizando adecuadamente las puertas XOR podemos diseñar un sumador/restador, de tal modo que si la línea auxiliar MODO = 0 suma, y si MODO = 1 resta. La figura 6-20 implementa un sumador/restador.

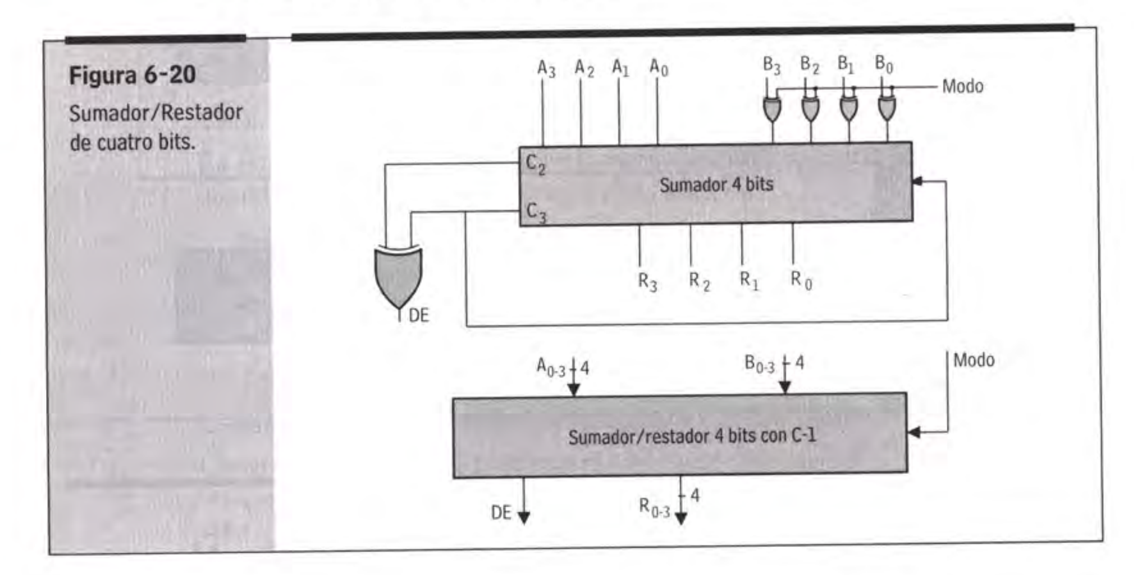


Figura 6-20
Sumador/Restador de cuatro bits.

Si MODO = 0 la operación que se plantea es R = A + B, mientras que si MODO = 1 la operación que se plantea es R = A + C-1(B), y por tanto el circuito ora se comporta como un sumador, ora como un restador.

Para finalizar planteemos cuatro ejemplos: dos sin desborde y dos con desborde. Queda para el lector el ejercicio (-0) - (+7), en el que es interesante observar el comportamiento de su desborde.

Ejemplo 6-1

Restar (+4) - (+2), (+3) - (+5), (+4) - (-6) y (-5) - (+5)

	1	1	0	0	0
(+4)		0	1	0	0
(-2)	+	1	1	0	1
	1	0	0	0	1
				+	1
(+2)	1	0	0	1	0
	DE = 1 ⊕ 1 = 0				

	0	0	1	0	0
(+3)		0	0	1	1
(-5)	+	1	0	1	0
	0	1	1	0	1
			+	0	
(-2)	0	1	1	0	1
	DE = 0 ⊕ 0 = 0				

Ejemplo 6-1 (Cont.)

	0	1	1	0	0
(+4)		0	1	0	0
(+6)	+	0	1	1	0
	0	1	0	1	0
				+	0
(-5)	0	1	0	1	0
	DE= $0 \oplus 1 = 1$				

	1	0	1	0	0
(-5)		1	0	1	0
(-5)	+	1	0	1	0
	1	0	1	0	0
				+	1
(+5)	1	0	1	0	1
	DE= $1 \oplus 0 = 1$				

6.6.2. Sumador y restador en binario puro con C-2

Si el código elegido es el binario puro con C-2, el enfoque es idéntico al correspondiente al binario puro con C-1. Así, el algoritmo queda:

- Tomar A (minuendo) y B (sustraendo).
- Complementar a 2 (negar) el sustraendo B' = C-2 (B).
- Obtener la resta como R = A + B', sin recircular el acarreo.

Estudio del desborde

Se puede repetir el planteamiento correspondiente al complemento a 1 para obtener la expresión de cuándo hay desbordamiento en el código.

$$\text{Desborde} = C_3 \oplus C_2 = C_{n-1} \oplus C_{n-2},$$
$$\text{donde n} = \text{n}^{\text{o}} \text{ de bits} \qquad (6.14)$$

Para implementar el restador recordemos que el complemento a 2 de un número es su complemento a 1 más 1, C-2 = C-1+1, así el circuito resultante es el de la figura 6-21.

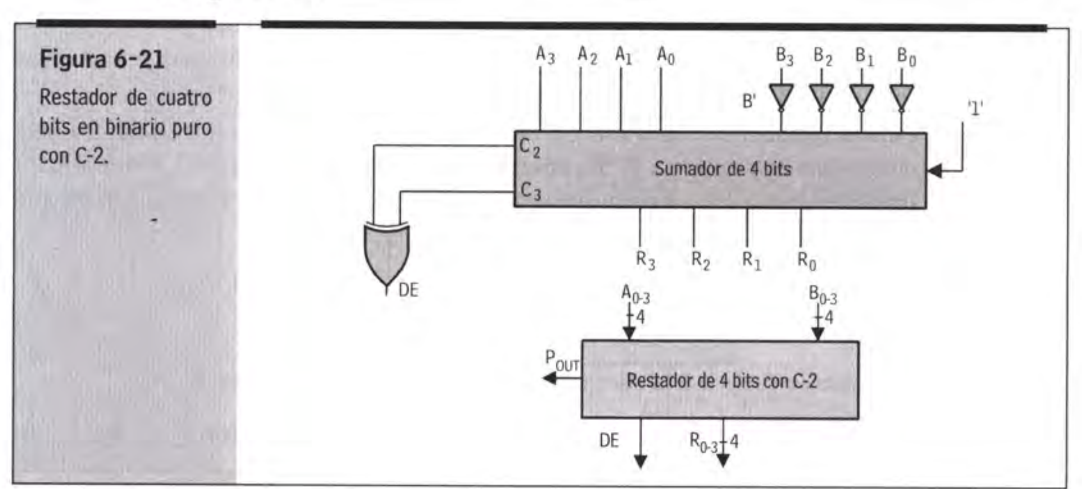


Figura 6-21
Restador de cuatro bits en binario puro con C-2.

En este caso la extensión de la capacidad de un restador es directa, sin más que duplicar el circuito correspondiente al restador de 4 bits y unirlo con el original.

El diseño del sumador/restador en este código es muy sencillo, sólo hay que tener en cuenta que si MODO = 1 (resta) hay que C-2 el operando B, y que si MODO = 0 (suma) no hay que complementar. Así, un sumador/restador en binario puro con C-2 para 8 bits queda como muestra la figura 6-22.

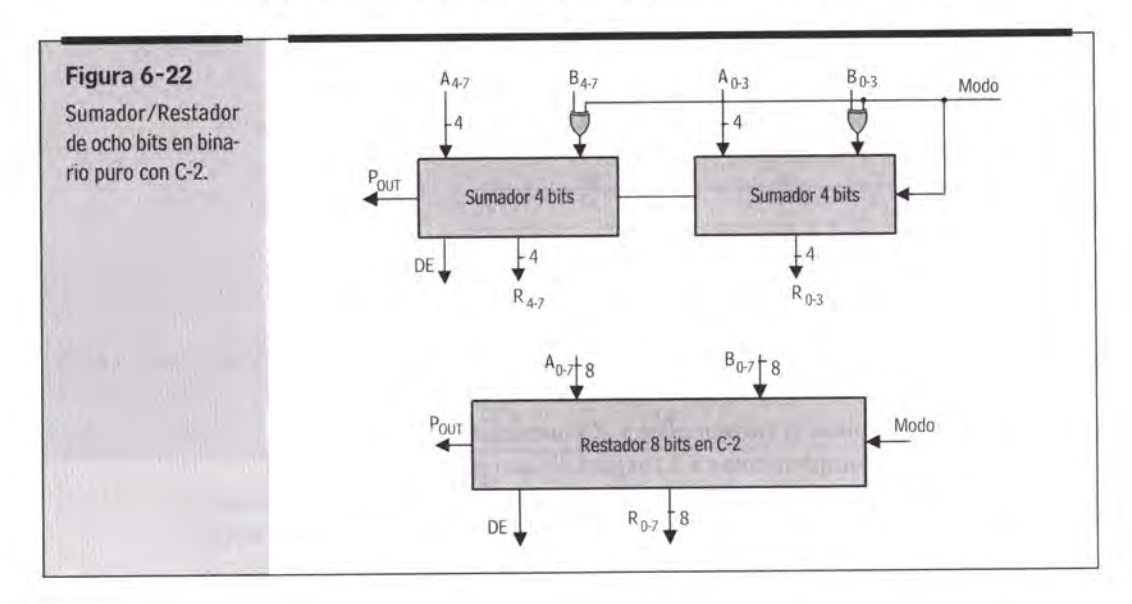


Figura 6-22
Sumador/Restador de ocho bits en binario puro con C-2.

Si MODO = 0: R = A + B

Si MODO = 1: R = A + C-1(B) + 1 = A-B.

6.6.3. Sumador y restador en binario con exceso

Al número decimal a codificar se le suma un exceso, codificándose el resultado de dicha suma en binario puro. En principio el exceso puede ser cualquier valor, pero los valores más normales son : 2^{n-1} ó 2^{n-1}-1, donde n es el número de bits asociados al código. Por ejemplo, para cuatro bits el exceso puede ser +8 o +7 y para 12 bits 2048 o 2047.

Para desarrollar un sumador con exceso, observemos la suma de dos números de cuatro bits con exceso a 7. Si sumamos dos números con exceso a 7 el resultado tendrá exceso a 14, así para que sea en exceso a 7 habrá que restarle 7, o, lo que es lo mismo, sumarle 9:

$$A(XS7) + B(XS7) = S(XS14) \qquad (6.15)$$
$$S(XS7) = S(XS14)- 7 = S(XS14) + 9$$

Si el exceso fuese a 8 la suma respondería a la siguiente expresión:

$$A(XS8) + B(XS8) = S(XS16) \qquad (6.16)$$
$$S(XS8) = S(XS16)-8 = S(XS16) + 8$$

 © ITES - PARANINFO

Estudio de desborde

El posible desborde se obtiene a partir de los acarreos obtenidos por el sumador principal (aquel que suma A y B) y por el sumador corrector (aquel que suma +8 o +9). El estudio en detalle del desborde es muy parecido a los anteriores y queda en manos del lector. La expresión resultante es:

$$\text{Desborde} = \overline{(C_{OUT}P \oplus C_{OUT}C)} \qquad (6.17)$$

$C_{OUT}P$: acarreo del sumador principal,

$C_{OUT}C$: acarreo del sumador corrector

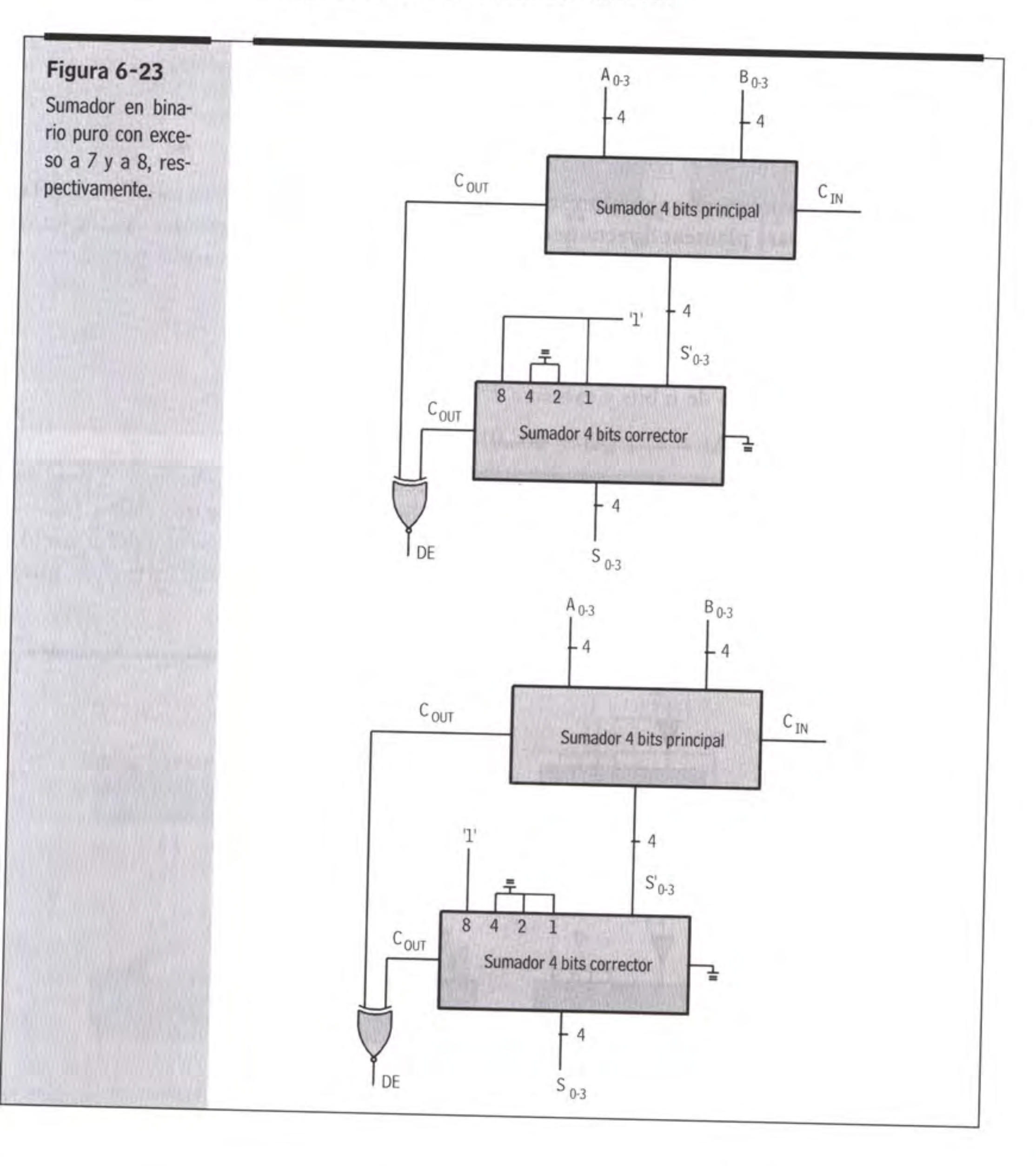


Figura 6-23
Sumador en binario puro con exceso a 7 y a 8, respectivamente.

A simple vista, en la figura 6-23 puede parecer que ambos circuitos son directamente extensibles puesto que disponen de C_{IN} y C_{OUT}, pero no es así, ya que la etapa correctora varía con el número de bits de una forma no extensible.

Algoritmo

Para extender la capacidad detallemos el algoritmo de un sumador de n bits con exceso a 2^{n-1} ó 2^{n-1}-1:

1. Tomar A y B y sumarlos mediante sumadores de 4 bits.

2.a Si el exceso es 2^{n-1} sumar a la última etapa 1000 y al resto 0000.

$$A(XS) + B(XS) = S(2XS) + 10.....000$$

2.b. Si el exceso es 2^{n-1}-1 sumar a la última etapa 1000, a la primera 0001 y al resto 0000.

$$A(XS) + B(XS) = S(2XS) + 10.....001$$

3. Establecer el posible desbordamiento.

No plantearemos ahora un sumador de más de una etapa, antes estudiaremos la resta para plantear directamente el circuito de un sumador/restador. Analicemos la resta empezando por saber cómo negar en códigos con exceso:

- Código de n bits y exceso a 2^{n-1} .

$$-(A) = C\text{-}1\ (A) + 11.....1\ (n\ bits)$$

- Código de n bits y exceso a 2^{n-1}

$$-(A) = C\text{-}1\ (A) + 00...01\ (n\ bits) = C\text{-}2(A)$$

Parece más cómodo y económico implementar el código con exceso 2^{n-1}, pero en la figura 6-24 vemos que para ambos códigos la negación de tres etapas consume tres sumadores (en el caso de 2^{n-1}-1 para permitir la transmisión del acarreo), y que además la negación en el código 2^{n-1}-1 es más fácilmente extensible, pues siempre es igual.

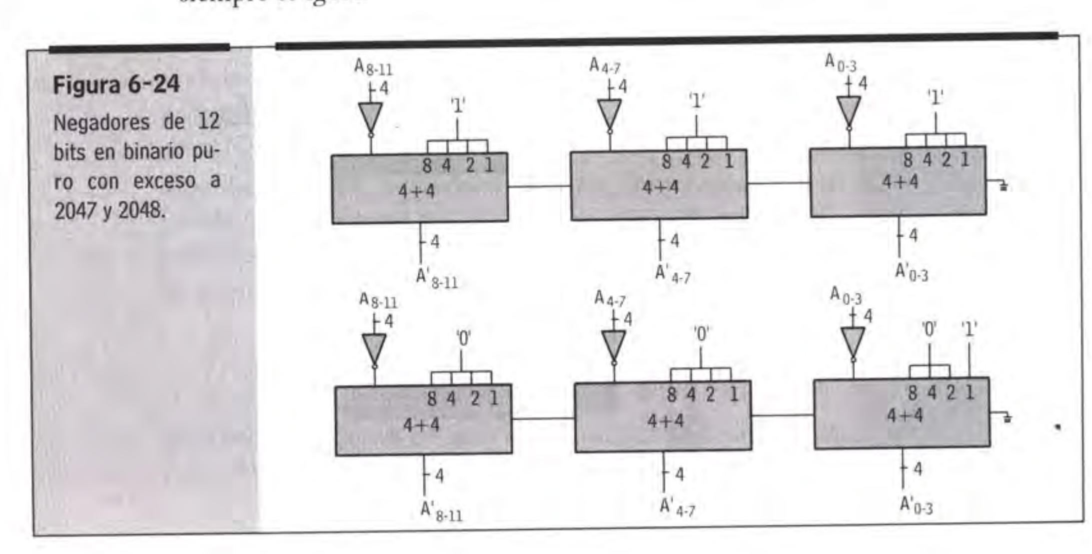


Figura 6-24
Negadores de 12 bits en binario puro con exceso a 2047 y 2048.

En las figuras 6-25 y 6-26 aparecen el sumador/restador de 12 bits y exceso 2047 y el correspondiente a un exceso 2048.

Figura 6-25
Sumador/Restador de 12 bits en binario puro con exceso a 2047.

Figura 6-26
Sumador/Restador de 12 bits en binario puro con exceso a 2048.

Ejemplo 6-2

Sumar en XS7 de 3+4 y (-6)+(-3) y restar 3-5 y 3-(-5) en XS8.

(+3)		1	0	1	0
(+4)	+	1	0	1	1
	1	0	1	0	1
	+	1	0	0	1
(+7)	*0*	1	1	1	0

$DE=\overline{(1 \oplus 0)} = 0$

(-6)		0	0	0	1
(-3)	+	0	1	0	0
	0	0	1	0	1
	+	1	0	0	1
(+7)	*0*	1	1	1	0

$DE=\overline{(0 \oplus 0)} = 1$

(+3)		1	0	1	1
(-5)	+	0	0	1	1
	0	1	1	1	0
	+	1	0	0	0
(-2)	*1*	0	1	1	0

$DE=\overline{(0 \oplus 1)} = 0$

(+3)		1	0	1	1
(+5)	+	1	1	0	1
	1	1	0	0	0
	+	1	0	0	0
(-7)	*1*	0	0	0	0

$DE=\overline{(1 \oplus 1)} = 1$

6.7. Sumador y restador en códigos BCD

En el anterior apartado hemos implementado sumadores y/o restadores basados en binario puro con signo, es decir, en códigos de palabra. En este apartado el objetivo es desarrollar idénticos circuitos para códigos BCD:

- BCD puro sin signo.
- BCD puro con signo en C-9.
- BCD puro con signo en C-10.
- BCD XS3 sin signo.
- BCD XS3 con signo en C-9.

Obtendremos para cada uno de los códigos anteriores un circuito sumador, un circuito negador, el correspondiente circuito restador y el circuito sumador/restador. Seguiremos en cada circuito una pauta:

- Descripción del código.
- Determinación del algoritmo de la operación.
- Determinación de la situación de desborde.
- Implementación del circuito.

- Estudio de la extensibilidad del circuito.
- Ejemplo aclarativo si fuera necesario.

6.7.1. Sumador BCD

El código BCD asocia a cada dígito decimal cuatro bits. Al sumar dos dígitos A y B en BCD el resultado puede ser mayor o menor que diez. Si el resultado fuera menor que diez sería correcto, no así si fuera mayor que diez. Por ejemplo, si sumamos 7 + 5 resulta 0111 + 0101 = 1100, que no es 12 en BCD.

Así pues, si sumáramos A+B obtendríamos un resultado S' y un C'_{OUT} que no tendrían por qué ser correctos, y que por tanto habría que modificar para que fueran los adecuados.

A continuación se clasifican los posibles resultados de la suma, indicando si la suma es correcta o no, cómo determinar su incorrección y cómo corregirla. También se contempla el comportamiento del C_{OUT}.

- $A+B \leq 9$. El resultado de la suma y el acarreo son correctos.
- $10 \leq A+B \leq 15$. El resultado de la suma y el acarreo no son correctos.
- $A+B \geq 16$. El resultado de la suma no es correcto, el del acarreo sí.

Generalizando, cuando la suma es mayor o igual que 10 hay que transformar el resultado, restándole 10, o lo que es lo mismo, sumándole 6.

Ya sabemos qué hacer y cuándo hacerlo, pero nos falta expresarlo en términos booleanos. El resultado de sumar A+B es mayor que diez:

$$S'>10 = C'_{OUT} + S'_3S'_2 + S'_3S'_1$$

Si reescribimos lo anterior en forma de expresión booleana, donde (A + B) BP significa sumar A y B en binario puro y (A + B) BCD lo correspondiente en BCD, el resultado es el siguiente:

$$S' = (A + B)\ BP$$

$$S'>10 = C'_{OUT} + S'_3S'_2 + S'_3S'_1$$

$$S = S' + (0110) \cdot (S'>10) = (A + B)\ BCD \qquad (6.18)$$

$$C_{OUT} = (S'>10)$$

Expresado algorítmicamente:

- Tomar A y B como sumandos.
- *Etapa sumadora.* Sumar A y B en un sumador de binario puro.

$$C'_{OUT}\ S'_3\ S'_2\ S'_1\ S'_0 = A_3A_2A_1A_0 + B_3B_2B_1B_0$$

- *Etapa correctora.* Corregir S' y C_{OUT} si el número es ≥ 10.

$$S'>10 = S'_3\ S'_2 + S'_3\ S'_1 + C'_{OUT}$$

$$S_3\ S_2\ S_1\ S_0 = S'_3\ S'_2\ S'_1\ S'_0 + (0110) \cdot (S'>10)$$

$$C_{OUT} = C'_{OUT} + S'_3\ S'_2 + S'_3\ S'_1 = (S'>10)$$

El esquema correspondiente al anterior algoritmo es el de la figura 6-27.

Figura 6-27
Sumador BCD de un dígito o etapa.

Observando el circuito vemos que el acarreo de la etapa correctora es despreciado, que el circuito es directamente extensible y que el desborde es indicado por el propio acarreo final.

En el esquema de la figura 6-28 se suman en BCD los números 18 + 34 = (0001 1000) + (0011 0100) = 52 (0101 0010).

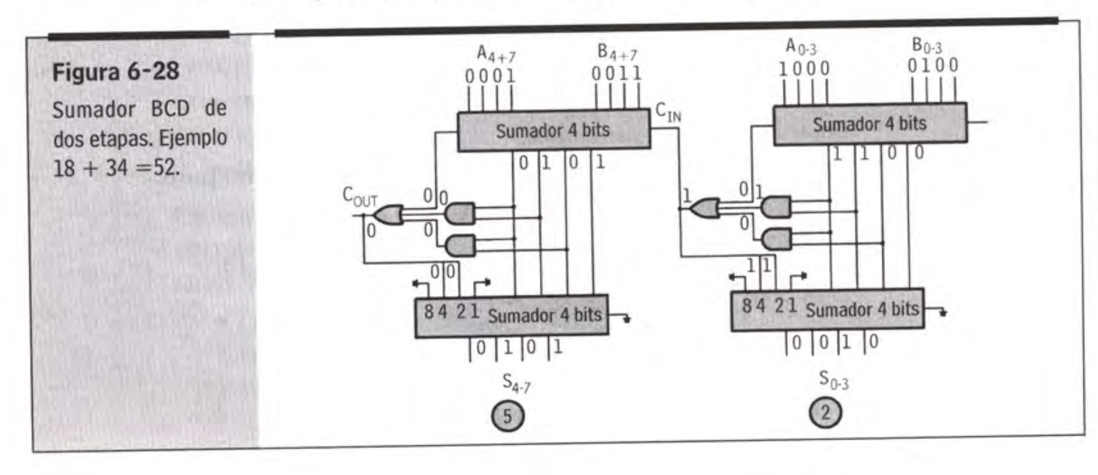


Figura 6-28
Sumador BCD de dos etapas. Ejemplo 18 + 34 =52.

6.7.2. Sumador y restador en BCD con signo en complemento

En los códigos BCD los complementos a 9 y a 10 cumplen las mismas funciones que los C-1 y C-2 de binario puro, y por tanto podrán ser aplicadas estrategias parejas.

En BCD con signo por C-9 o C-10 los números positivos se escriben como BCD precedidos por un 0 (0000), mientras que los números negativos se escriben como el C-9 o el C-10 del correspondiente número positivo.

Ejemplo 6-3

Sumar (+626) + (-285) utilizando el código decimal con C-9 y (+328) + (-825) utilizando el C-10.

```
+626 => 0626     0626
-285 => 9714   + 9714
+341            10340
                   +1
                 0341 => +341
+328 => 0328     0328
-825 => 9175   + 9175
-497             9503 =>  -497
```

Algoritmo de suma en BCD con C-9

1. Tomar A y B.
2. Sumar en BCD A+B, recirculando el acarreo final.
3. Estudiar el desborde.

Algoritmo de suma en BCD con C-10

1. Tomar A y B.
2. Sumar en BCD A+B, sin recircular el acarreo final.
3. Estudiar el desborde.

Estudio del desborde

El análisis del desborde sólo tiene sentido cuando ambos sumandos son positivos o negativos, pues sólo en este caso se puede producir desborde. El siguiente estudio es válido tanto para BCD con C-9 como para BCD con C-10. El número A se compone de dígito S_A para el signo y de M_A para la magnitud (complementada o no), lo mismo para B.

- A y B son positivos.

$$\text{Des. positivo} = \overline{C_{OUTsigno}} \cdot C_{OUTmag}$$

- A y B son negativos.

$$\text{Des. negativo} = C_{OUTsigno} \cdot \overline{C_{OUTmag}}$$

Reuniendo ambas condiciones de desborde resulta:

$$DE = \overline{C_{OUTmag}} \cdot C_{OUTsigno} + \overline{C_{OUTsigno}} \cdot C_{OUTmag} =$$
$$= C_{OUTsigno} \oplus C_{OUTmag}$$
$$DE = C_{OUTsigno} \oplus C_{OUTmag} = C_n \oplus C_{n-1},$$
$$\text{donde } n = n^{\circ} \text{ de etapas} \qquad (6.19)$$

6.7.2.1. Sumador BCD con C-9 o C-10

Planteemos los circuitos sumadores de las figuras 6-29 y 6-30 para números de tres dígitos en BCD con C-9 y C-10, respectivamente. Ambos circuitos se basan en el sumador BCD anteriormente descrito.

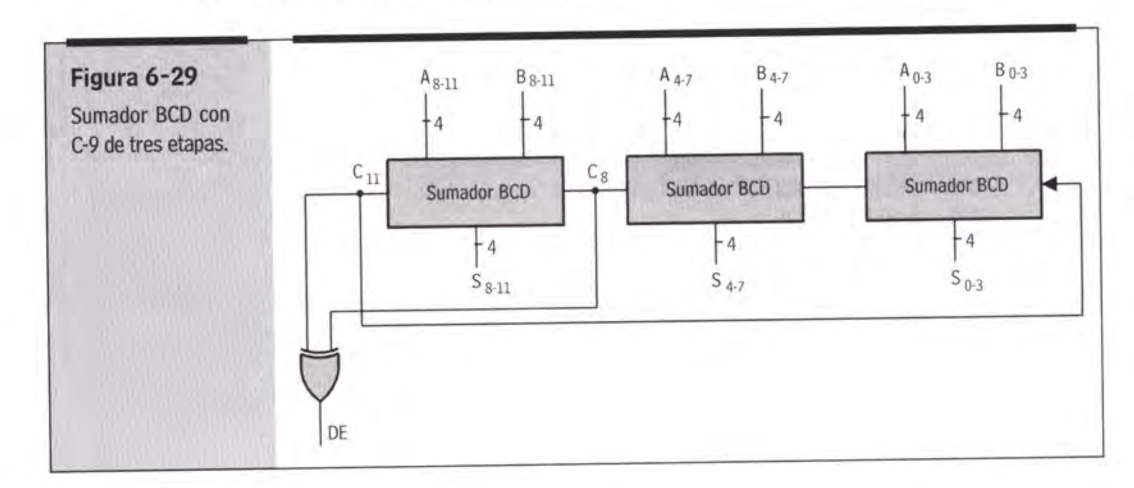


Figura 6-29
Sumador BCD con C-9 de tres etapas.

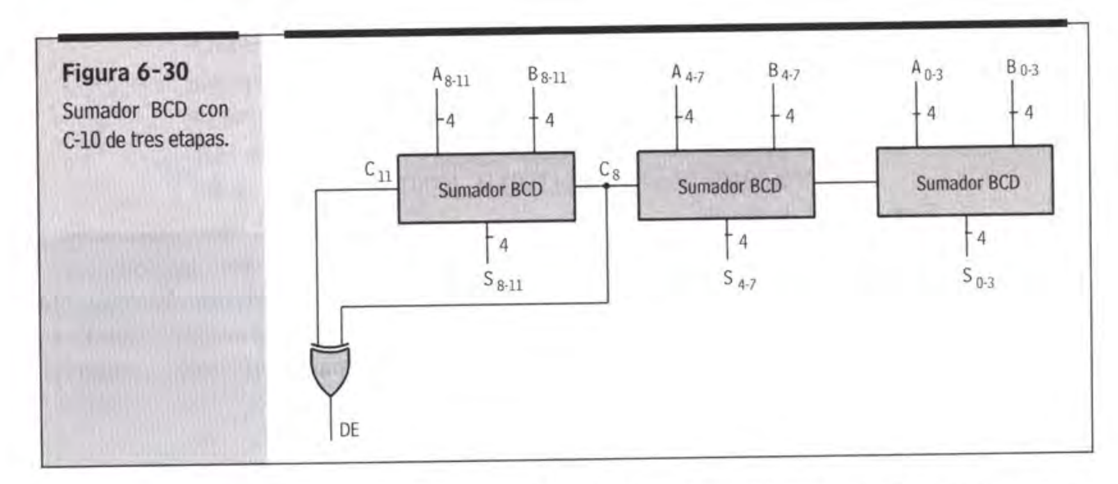


Figura 6-30
Sumador BCD con C-10 de tres etapas.

Los anteriores circuitos se pueden reducir si nos fijamos en que para un número positivo los cuatro bits de más peso son 0000, y que para un número negativo son 1001; por tanto, podríamos prescindir de los tres primeros bits, quedando el positivo como 0 y el negativo como 1. Así, un número de dos dígitos con signo tendría 8 bits para la magnitud y 1 para el signo, con el consiguiente ahorro, a

BIBLIOTECA CADIZ PUBLICA

costa de una pérdida de elegancia. La modificación del circuito es trivial, simplemente hay que sustituir el sumador BCD del signo por un sumador completo de 1 bit, FA. La figura 6-31 muestra un sumador BCD con C-10 para tres dígitos y signo, más económico que el anterior, pero con menor elegancia teórica.

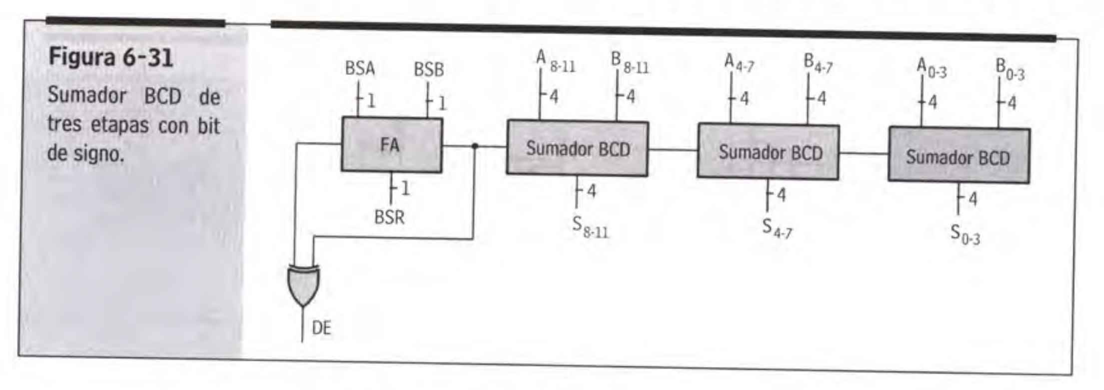


Figura 6-31
Sumador BCD de tres etapas con bit de signo.

6.7.2.2. Restador en BCD con C-9 o C-10

Una vez diseñado el circuito sumador con signo, diseñar el correspondiente restador pasa por implementar un circuito capaz de negar el sustraendo; y en ambos códigos negar es complementar, ya sea a 9 o a 10.

El complemento a 9 de cada dígito es el complemento a 1 más 10.

$$C\text{-}9(A) = C\text{-}1(A) + 1010 \qquad (6.20)$$

El circuito del C-9 de la figura 6-32 es directamente expandible para varios dígitos, ya que la obtención del C-9 de un dígito es independiente de los restantes dígitos.

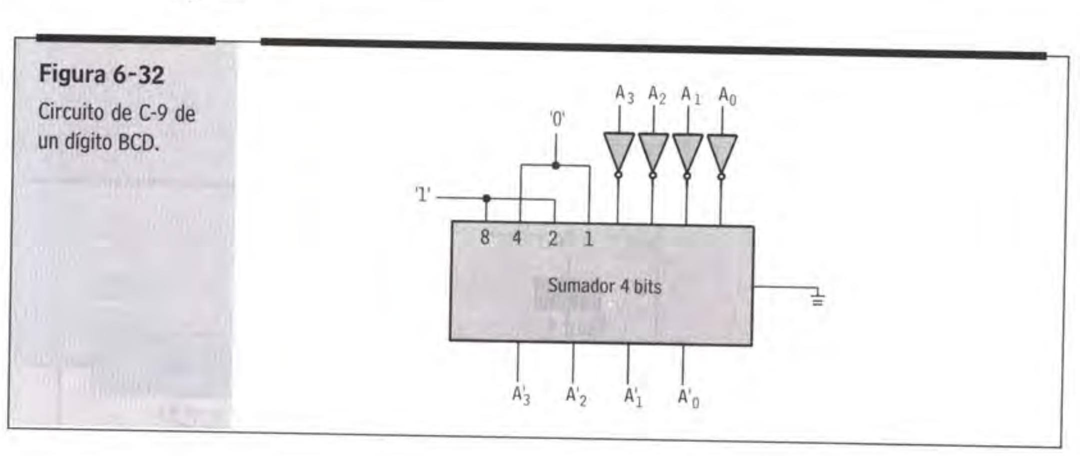


Figura 6-32
Circuito de C-9 de un dígito BCD.

El complemento a 10 de un número BCD al completo, no dígito a dígito, es el complemento a 9 de dicho número más 1. Este circuito no es expandible directamente como el anterior, además es más lento, pues debe tener en cuenta la propagación de acarreos entre etapas. Puede parecer que el C-10 es peor que el

C-9, pero recordemos que en el sumador con C-10 no había que recircular el acarreo final, y con C-9 sí. La figura 6-33 muestra un circuito de C-10 para tres dígitos.

$$\text{C-10(A)} = \text{C-1(A)} + 1010 + 1 \qquad (6.21)$$

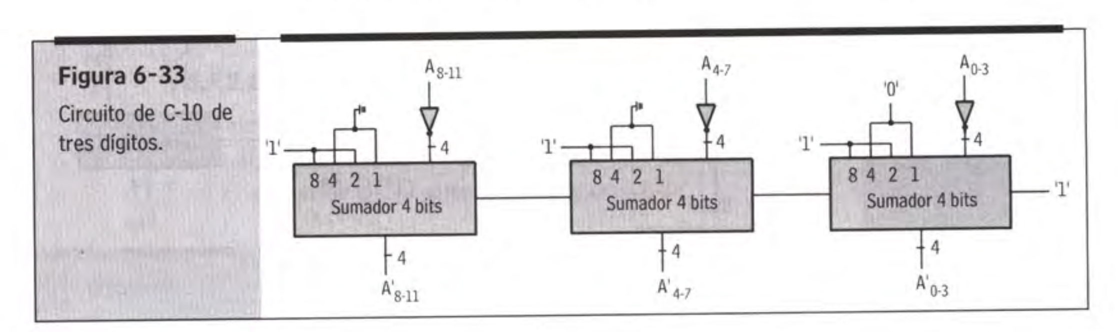


Figura 6-33
Circuito de C-10 de tres dígitos.

Los circuitos de las figuras 6-34 y 6-35 se corresponden con restadores en BCD con C-9 y C-10.

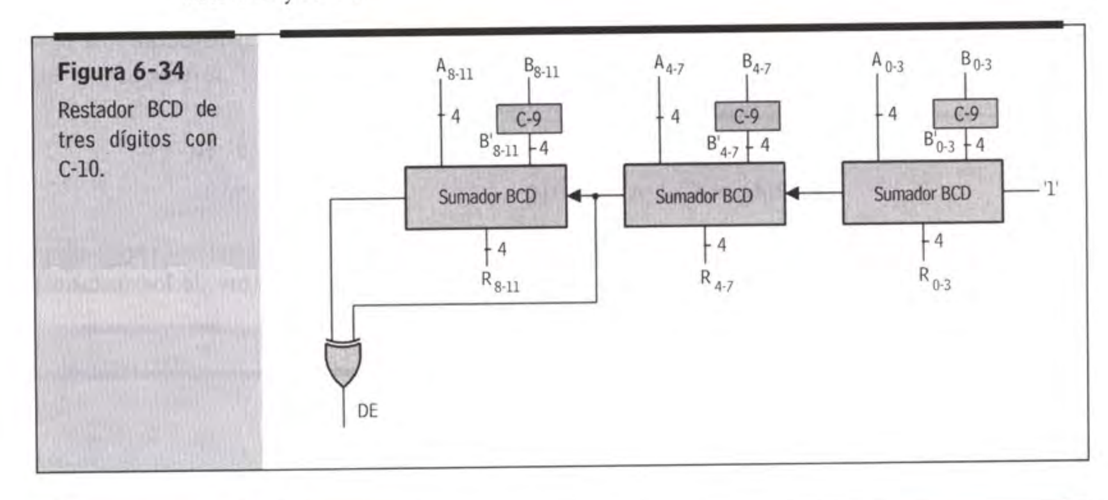


Figura 6-34
Restador BCD de tres dígitos con C-10.

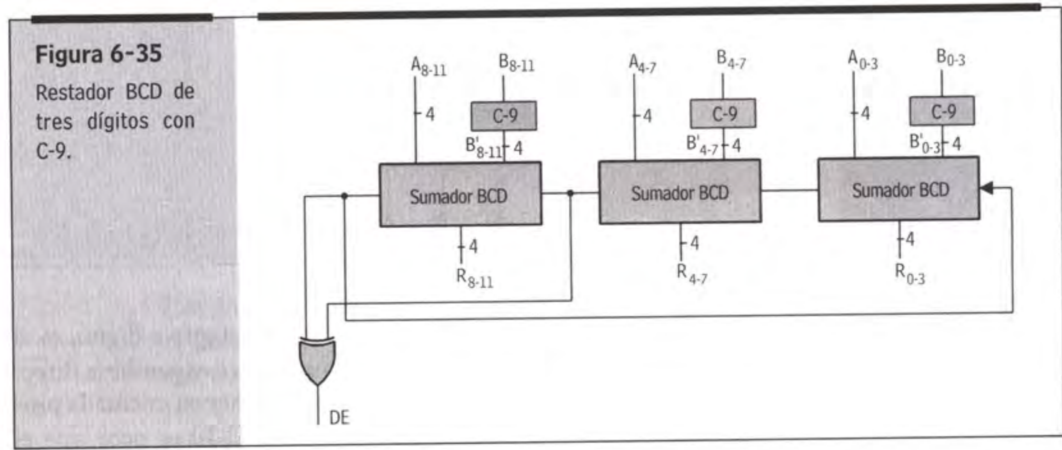


Figura 6-35
Restador BCD de tres dígitos con C-9.

6.7.2.3. Circuito sumador/restador

Los dos circuitos de las figuras 6-36 y 6-37 implementan sendos sumadores/restadores en BCD con C-9 y con C-10, respectivamente. El primero trabaja con bit de signo y tres dígitos BCD, y además utiliza un multiplexor para operar con suma o resta. En el caso del C-10 el circuito es un sumador/restador con un dígito de signo y dos de magnitud, sin multiplexores.

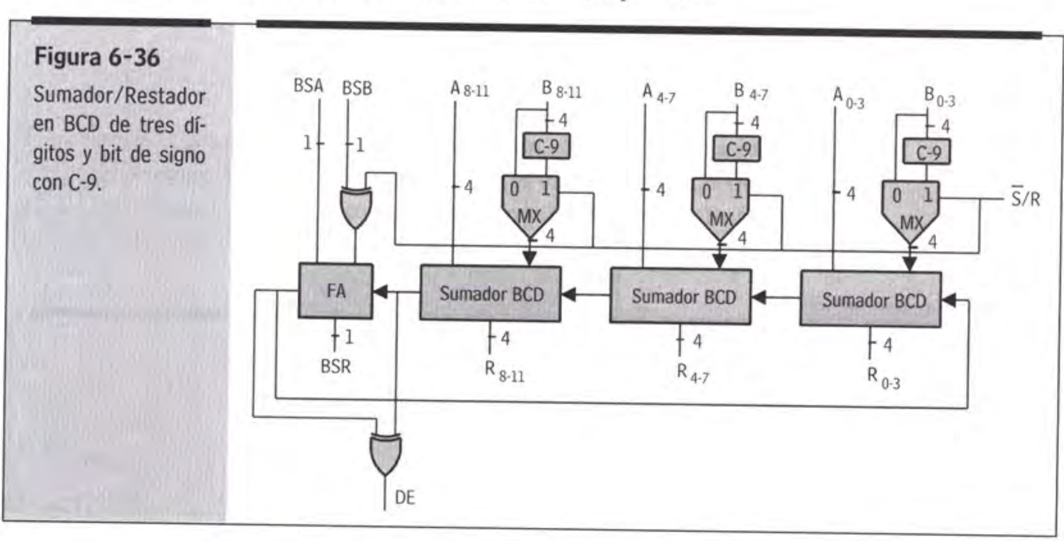


Figura 6-36
Sumador/Restador en BCD de tres dígitos y bit de signo con C-9.

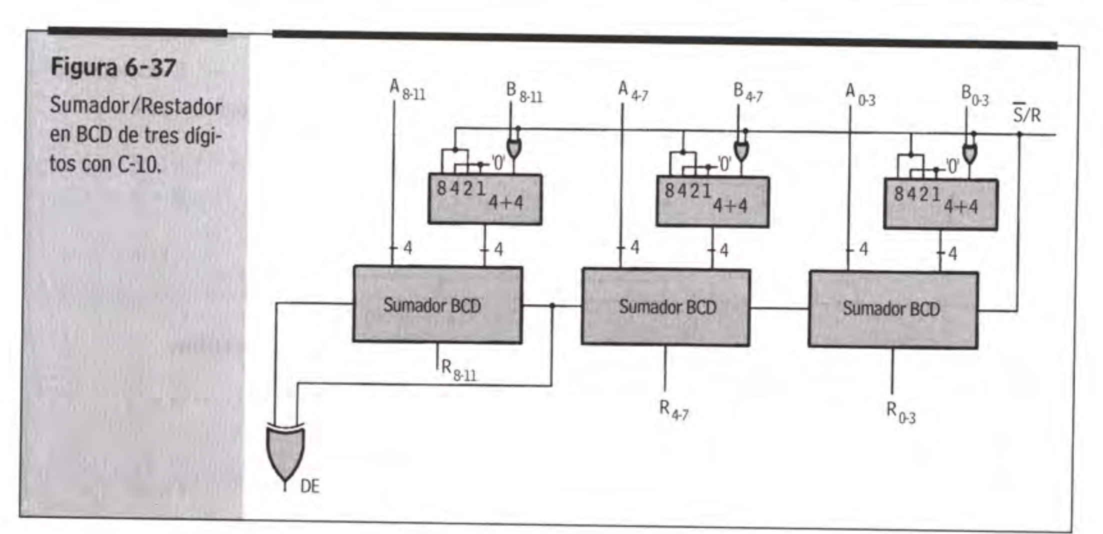


Figura 6-37
Sumador/Restador en BCD de tres dígitos con C-10.

6.7.3. Sumador en BCD XS3

El código BCD XS3 está íntimamente ligado al BCD puro, recordando que BCD XS3 = BCD+3. El BCD XS3 es autocomplementario, esto es, el C-9 de un número XS3 es igual a su complemento a 1: C-9(XS3) = C-1(XS3). La desventaja del XS3 frente al BCD puro es que no es un código próximo al hombre, pero tiene una clara ventaja: su cómoda aritmética.

Antes de describir el algoritmo, veamos cómo se desarrolla la suma:

- A(XS3) + B(XS3)= S(XS6), es decir, el resultado queda en XS6, luego habría que restarle 3 o sumarle 13.

  S(XS3) = S(XS6) - 3 = S(XS6) + 13

- El resultado, al igual que BCD puro, puede superar el valor 10, luego si el resultado fuera mayor o igual que 10 habría que corregirlo, restándole 10 o sumándole 3.

  S(XS3) = S(XS6) - 3 -10 = S(XS6) + 3

- En cuanto a la cuestión ¿cuándo es superior a 10 el resultado?, vemos que si un número es mayor que 10 el correspondiente XS6 será mayor o igual que 16, y por tanto se corresponderá con el propio C_{OUT} del sumador principal.

  CORR= S(XS6) ≥ 16 = C_{OUT}

Ejemplo 6-4

Estudio de cuándo la suma supera 10 en XS3: 8 + 3, 5 + 7, 9 + 8.

	1	0	1	1	8
	0	1	1	0	3
1	0	0	0	1	
	1	0	0	0	5
	1	0	1	0	7
1	0	0	1	0	
	1	1	0	0	9
	1	0	1	1	8
1	0	1	1	1	

Se ve que cuando la suma supera 10 se genera C_{OUT}.

Si reordenamos los anteriores puntos en forma algorítmica resulta:

- Tomar A y B en BCD XS3.
- Sumar A y B según el binario puro.

  S'(XS6) = (A + B) BP

- Si el resultado S'(XS6) < 10 => C_{OUT} = 0

  S(XS3) = S'(XS6)-3 = S'(XS6) + 13 = S'(XS6) + 1101

- Si el resultado S'(XS6) ≥ 10 => C_{OUT} = 1

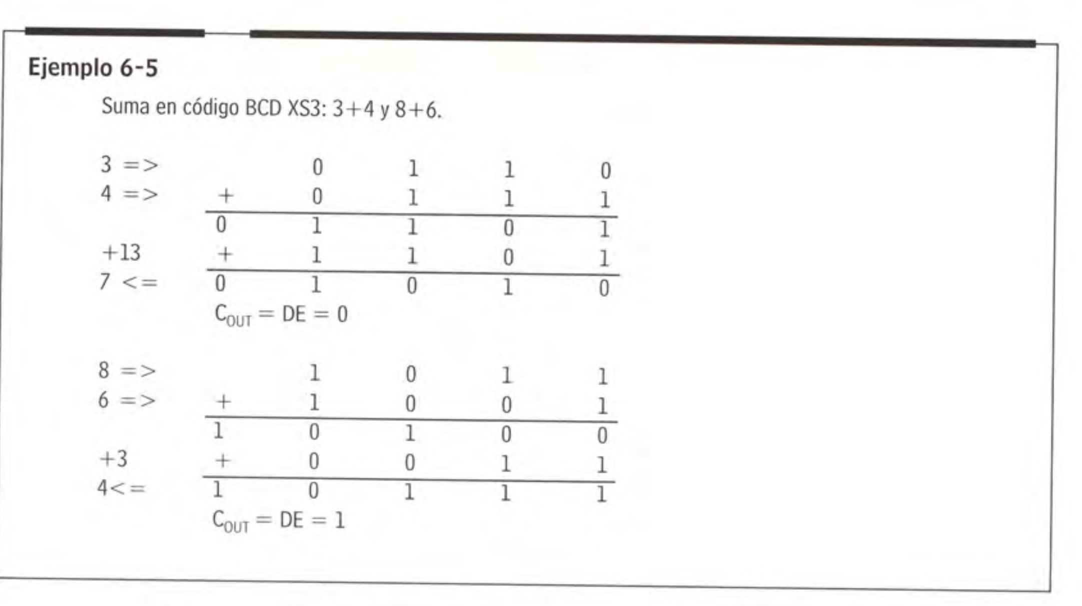
  S(XS3)=S'(XS6)-3-10 = S'(XS6) + 3 = S(XS6) + 0011

- El resultado es correcto y el desborde es indicado por el propio C_{OUT}.

La figura 6-38 se corresponde con los puntos anteriores.

Figura 6-38

Sumador BCD XS3 de un dígito o etapa.



Ejemplo 6-5

Suma en código BCD XS3: 3+4 y 8+6.

3 =>		0	1	1	0
4 =>	+	0	1	1	1
	0	1	1	0	1
+13	+	1	1	0	1
7 <=	0	1	0	1	0

$C_{OUT} = DE = 0$

8 =>		1	0	1	1
6 =>	+	1	0	0	1
	1	0	1	0	0
+3	+	0	0	1	1
4<=	1	0	1	1	1

$C_{OUT} = DE = 1$

Al comparar los circuitos de los sumadores BCD y XS3, resulta a simple vista que el sumador XS3 es más sencillo. Y lo es porque es más fácil determinar cuándo y cuánto hay que corregirlos. Esta ventaja para el XS3 se acrecienta si continuamos con el estudio del restador.

6.7.4. Sumador/Restador en BCD XS3 con signo en complemento

El planteamiento es muy rápido y sencillo, ya que el complemento a 9 de un número en XS3 coincide con su complemento a 1.

El algoritmo correspondiente es muy sencillo:

- Tomar A y B en BCD XS3.
- Negar el sustraendo B:

  -B = C-9(B) = C-1(B) = NOT(B).

- Sumar según el sumador en BCD XS3, recirculando el C_{OUT}.

  $R = (A + B)_{XS3} + C_{OUT}$

- Estudiar el desborde como:

  $D_E = C_{OUT} \oplus C_{n-1}$, C_{OUT} acarreo de la última etapa y C_{n-1} de la ante-última.

  $D_E = C_n \oplus C_{n-1}$, donde n es el nº de etapas (6.22)

Ejemplo 6-6

Restar en BCD XS3: (+35)-(32); (-12)-(+15); (-80)-(+60). Para seguir el ejemplo hay que recordar que las sumas son BCD XS3.

```
                      1       1       0
+35   =>   035          0 0 1 1 0 1 1 0 1 0 0 0
-32   =>   967          1 1 0 0 1 0 0 1 1 0 1 0
                      -------------------------
                      1 0 0 1 1 0 0 1 1 0 1 0 1
                                            + 1
                      -------------------------
+03   <=   003          0 0 1 1 0 0 1 1 0 1 1 0
                      DE = 1⊕1 = 0

                      1       1
-12   =>   987          1 1 0 0 1 0 1 1 1 0 1 0
-15   =>   984          1 1 0 0 1 0 1 1 0 1 1 1
                      -------------------------
                      1 1 1 0 0 1 0 1 0 0 1 0 0
                                            + 1
                      -------------------------
-27   <=   972          1 1 0 0 1 0 1 0 0 1 0 1
                      DE = 1⊕1 = 0

                      1       0
-80   =>   919          1 1 0 0 0 1 0 0 1 1 0 0
-60   =>   939          1 1 0 0 0 1 1 0 1 1 0 0
                      -------------------------
                      1 1 0 1 1 1 0 0 0 1 0 1 1
                                            + 1
                      -------------------------
???   <=   859          1 0 1 1 1 0 0 0 1 1 0 0
                      DE = 0⊕1 = 1
```

El circuito de la figura 6-39 representa a un restador en XS3 con C-9 para un número con dos dígitos de magnitud y uno de signo.

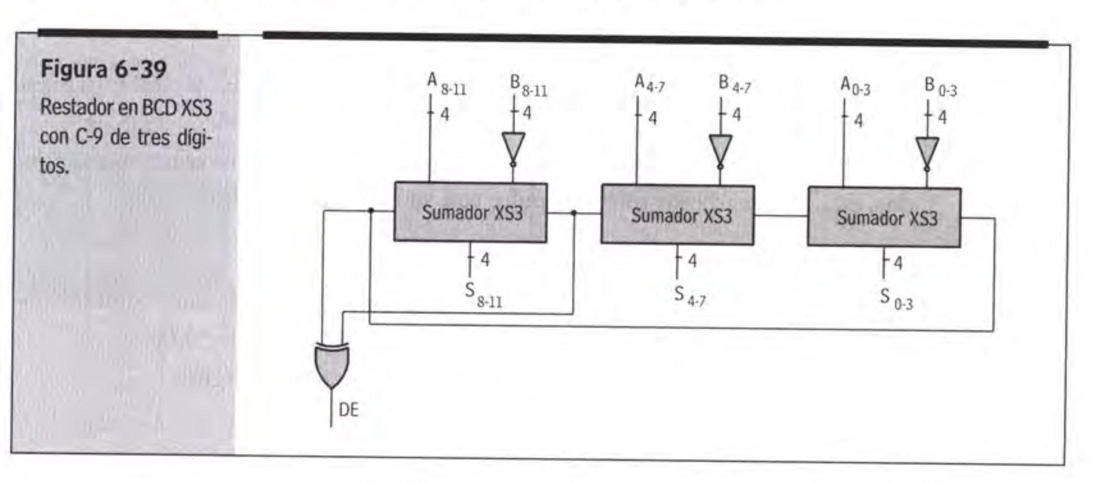


Figura 6-39
Restador en BCD XS3 con C-9 de tres dígitos.

Si planteáramos un sumador/restador el circuito sería el de la figura 6-40.

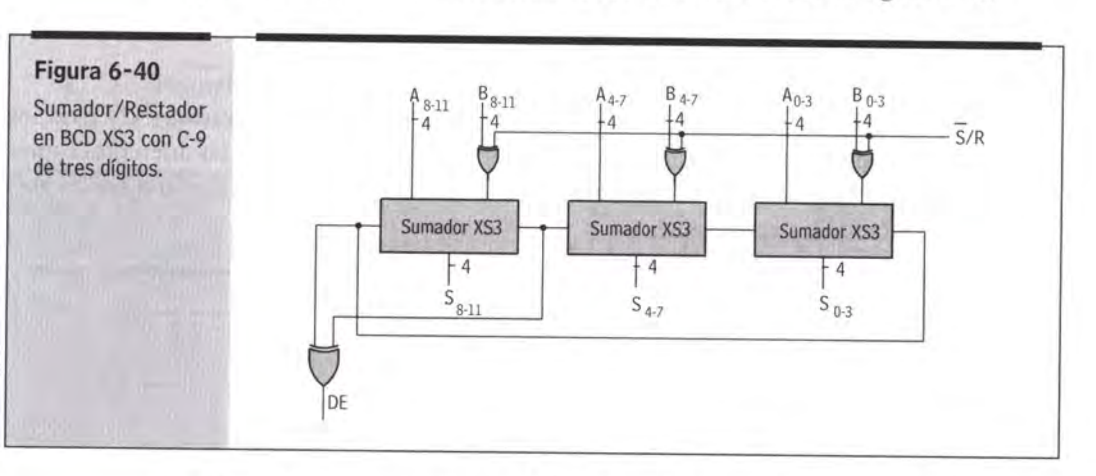


Figura 6-40
Sumador/Restador en BCD XS3 con C-9 de tres dígitos.

En el circuito correspondiente al C-10 simplemente deberíamos introducir un ´1´ por el acarreo de entrada del primer sumador XS3. Dejamos para el lector el código XS3 donde el signo no es un dígito, sino un solo bit.

6.8. Multiplicadores y divisores

Al igual que existen circuitos capaces de sumar y restar en diferentes códigos, también los hay capaces de multiplicar y dividir.

A la hora de implementar un multiplicador o divisor podemos plantear dos estrategias:

- La multiplicación es una suma iterada, y la división es una resta iterada.
- Multiplicar o dividir es conocer la correspondiente tabla de multiplicar o dividir.

La primera estrategia es independiente del número de bits de los operandos, mientras que la segunda es totalmente dependiente de dicho número de bits; la tabla cambiará con el número de bits. Además, la primera estrategia es secuencial (capítulo 7 y siguientes) y la segunda combinacional.

Los circuitos resultantes exceden en ocasiones el nivel del libro. Es preferible -a nuestro juicio- mantener el nivel de los contenidos, aunque sea a costa de reducirlos.

De todos los multiplicadores posibles: por acumulación, con contadores, rápidos, etc., plantearemos sólo el rápido, con su estrategia combinacional.

6.8.1. Multiplicador combinacional o paralelo o rápido

En este punto se contempla la multiplicación como una operación combinacional que responde a la manera clásica de multiplicar.

Desarrollemos una multiplicación de dos números de tres dígitos:

$$A_2\,A_1\,A_0 \times B_2\,B_1\,B_0 = P_5\,P_4\,P_3\,P_2\,P_1\,P_0$$

			A_2	A_1	A_0
			B_2	B_1	B_0
			A_2B_0	A_1B_0	A_0B_0
		A_2B_1	A_1B_1	A_0B_1	
	A_2B_2	A_1B_2	A_0B_2		
P_5	P_4	P_3	P_2	P_1	P_0

Podemos implementar la anterior operación utilizando puertas AND para los productos $A_i \times B_i$, y sumadores completos para ir obteniendo las diferentes sumas concatenadas. En el diseño de la figura 6-41 hay que tener cuidado con la gestión de los acarreos.

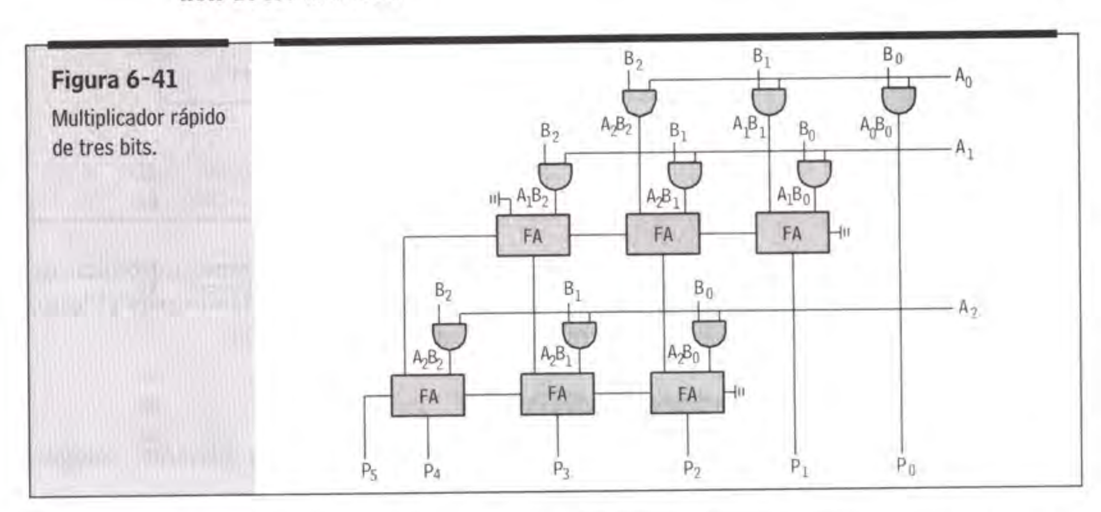


Figura 6-41
Multiplicador rápido de tres bits.

Antes de plantear un multiplicador de 4 bits nos será muy útil presentar el circuito UBMC (Unidad Básica del Multiplicador Combinacional) en la figura 6-42. Este circuito UBMC multiplica dos bits y suma este resultado a otro procedente de un UBMC, es decir, permite la conexión entre UBMC a través del C_{i-1} y C_i.

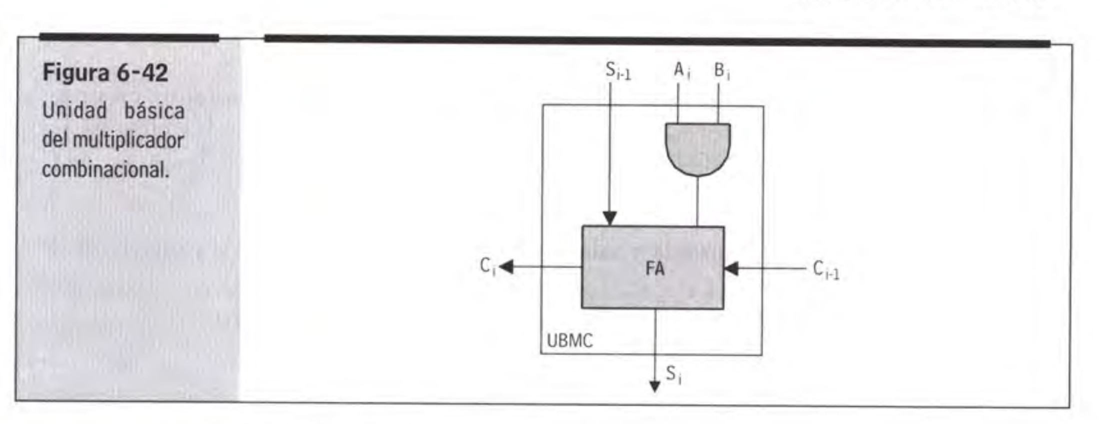
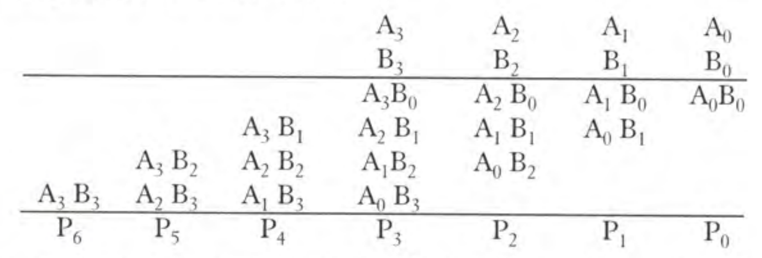


Figura 6-42
Unidad básica del multiplicador combinacional.

Una multiplicación de cuatro bits es:



			A_3	A_2	A_1	A_0
			B_3	B_2	B_1	B_0
			A_3B_0	$A_2 B_0$	$A_1 B_0$	A_0B_0
		$A_3 B_1$	$A_2 B_1$	$A_1 B_1$	$A_0 B_1$	
	$A_3 B_2$	$A_2 B_2$	A_1B_2	$A_0 B_2$		
$A_3 B_3$	$A_2 B_3$	$A_1 B_3$	$A_0 B_3$			
P_6	P_5	P_4	P_3	P_2	P_1	P_0

Planteemos ahora en la figura 6-43 el multiplicador de 4 bits utilizando como elemento básico el UBMC.

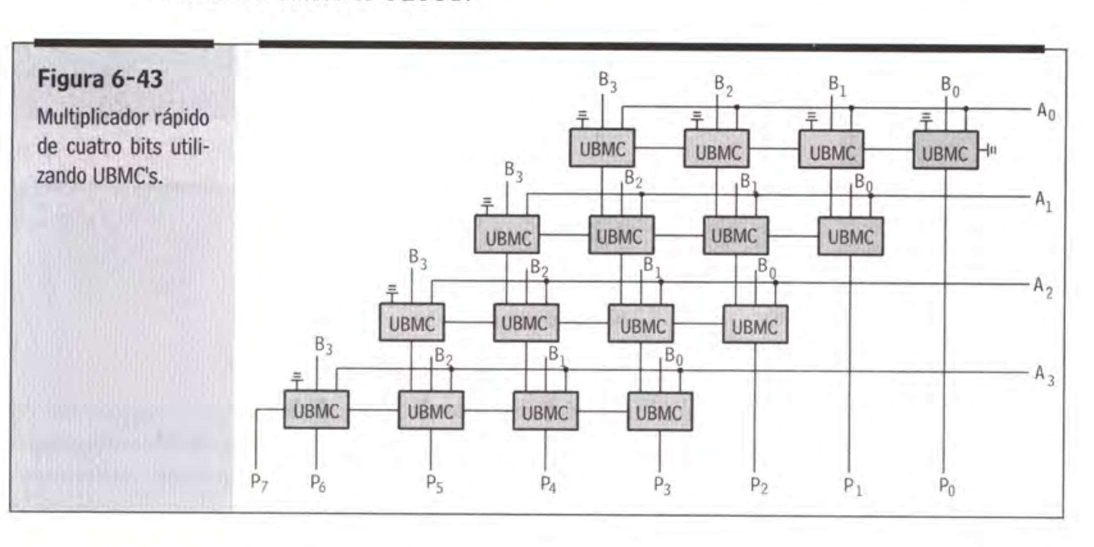


Figura 6-43
Multiplicador rápido de cuatro bits utilizando UBMC's.

Este mismo planteamiento puede ser implementado obteniendo en un primer bloque todos los productos mediante la matriz de puertas AND (gate array), que, en un segundo bloque formado por sumadores de 4 bits (con acarreo serie o anticipado), serán sumados para obtener el producto. O también, obteniendo los acarreos y las sumas de pares de productos por separado, para luego sumarlos mediante sumadores de cuatro bits.

6.9. ALU's y circuitos MSI

Las unidades aritmético-lógicas son circuitos tipo MSI. Por tanto, en un solo circuito integrado disponemos de diversas operaciones, todas ellas básicas. En una UAL (Unidad Aritmético Lógica, ALU en inglés) hay señales de tres tipos:

- Líneas de entrada y salida. Son los operandos de la UAL, suelen ser de 4 bits.
- Líneas de control. Indican qué operación se va a realizar.
- Líneas auxiliares.

Las operaciones más comunes implementadas son la suma, la resta, el complemento a 1 o a 2 y el producto, la suma y la negación lógica. Cada circuito integrado implementa distintas funciones.

En la tabla 6-7 se describe el 74381 con entradas A y B de cuatro bits, donde las salidas toman su valor según las líneas de control S2, S1 y S0. Además dispone de las salidas auxiliares P, G y Cout para facilitar la interconexión de varios 74381.

Tabla 6-7
Funcionamiento del 74381.

S2	S1	S0	Q3Q2Q1Q0
0	0	0	Clear
0	0	1	B - A
0	1	0	A - B
0	1	1	A plus B [1]
1	0	0	$A \oplus B$
1	0	1	A + B
1	1	0	A · B
1	1	1	Preset

[1] Suma aritmética

El 74181 descrito en la tabla 6-8 dispone de 6 líneas de control -S3, S2, S1, S0, CY y BA- para operar con las entradas A y B de cuatro bits, y así obtener la salida Q -entradas y salidas son activas por nivel bajo-. Dispone también de líneas P, G y Cout que facilitan la interconexión de varios 74181.

Además de las ALU descritas, en la tabla 6-9 se resumen los principales circuitos aritméticos MSI de la serie 74.

Tabla 6-8
Tabla de funcionamiento del 74181.

				BA = 1 Lógica	BA = 0 Aritmética	
S3	S2	S1	S0	CY = X	CY = 1	CY = 0
0	0	0	0	$\overline{A}$	A	A plus 1
0	0	0	1	$\overline{A+B}$	A+B	(A+B) plus 1
0	0	1	0	$\overline{A}$ B	A+$\overline{B}$	(A+$\overline{B}$) plus 1
0	0	1	1	0	-1	0
0	1	0	0	$\overline{AB}$	A plus (A$\overline{B}$)	A plus (A$\overline{B}$) plus 1
0	1	0	1	$\overline{B}$	(A+B) plus (A$\overline{B}$)	(A+B) plus (A$\overline{B}$) plus 1
0	1	1	0	A⊕B	A-B-1	A-B
0	1	1	1	A$\overline{B}$	(A$\overline{B}$)-1	A$\overline{B}$
1	0	0	0	$\overline{A}$ +B	A plus (A$\overline{B}$)	A plus (AB) plus 1
1	0	0	1	$\overline{A \oplus B}$	A plus B	A plus B plus 1
1	0	1	0	B	(A+$\overline{B}$) plus (AB)	(A+$\overline{B}$) plus (AB) plus 1
1	0	1	1	AB	(AB)-1	AB
1	1	0	0	1	A plus A	A plus A plus 1
1	1	0	1	A+$\overline{B}$	(A+B) plus A	(A+B) plus A plus 1
1	1	1	0	A+B	(A+$\overline{B}$) plus A	(A+$\overline{B}$) plus A plus 1
1	1	1	1	A	A-1	A

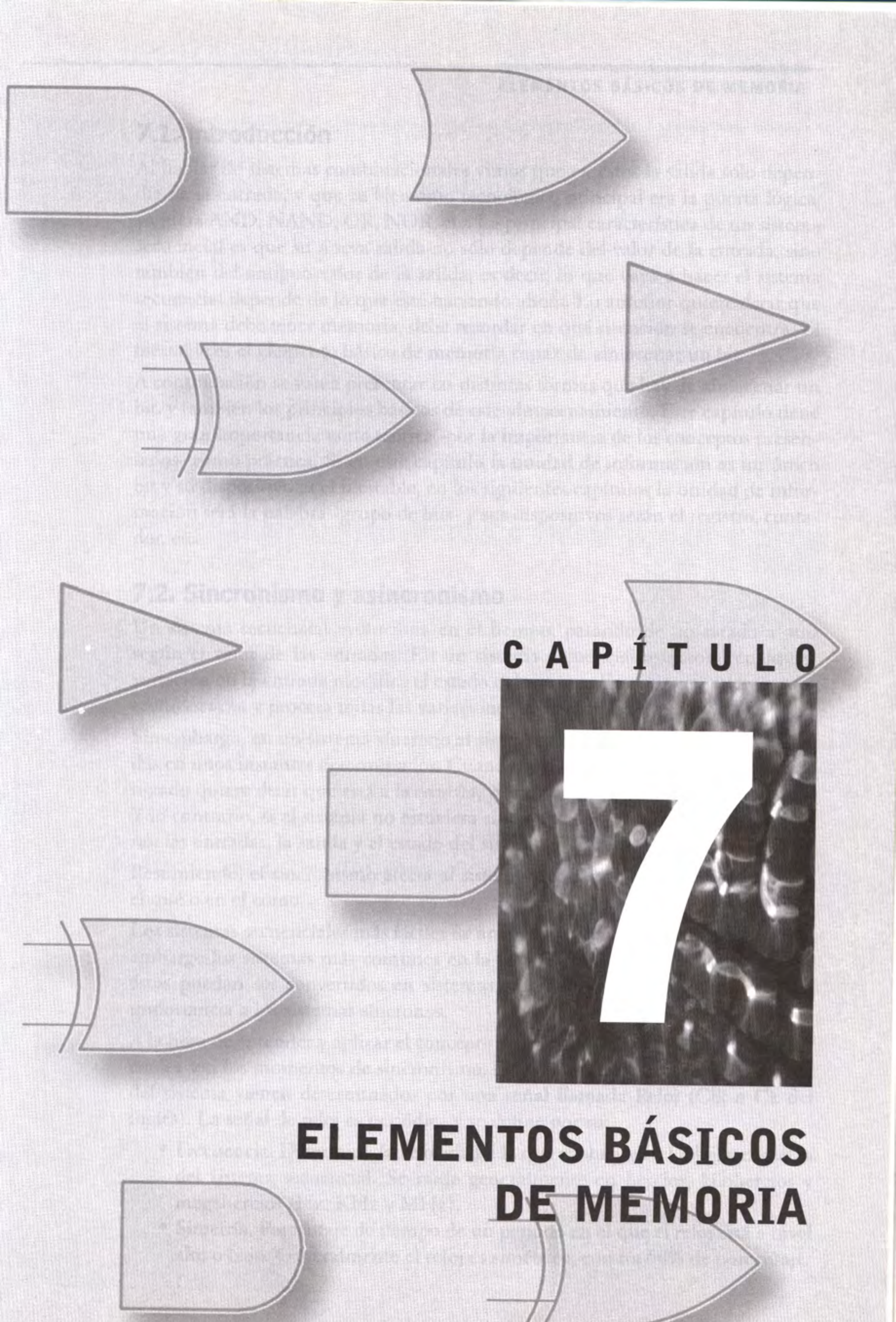
Tabla 6-9
Tabla de funcionamiento del 74181.

Dispositivo	Descripción	Tamaño	Entradas	Salidas	Líneas Auxiliares
74183	Sumador Completo	2 FA	Alto	Alto	
7483	Sumador binario	4 + 4	Alto	Alto	C_{IN} y C_{OUT}
74583	Sumador BCD	1 Dígito	Alto	Alto	C_{IN} y C_{OUT}
74182	Circuito LAC	4 bits	Bajo	Alto	P y G de grupo
74181	ALU	4 op 4	Bajo	Bajo	S3-0, BA, A=B, C_{IN} y C_{OUT}
74381	ALU	4 op 4	Alto	Alto	S2-0, C_{IN} y C_{OUT}

6.10. Resumen

En este capítulo se ha visto cómo la aritmética básica binaria es sencilla y múltiple. Sencilla porque se basa en la suma binaria de 1 bit y múltiple por la gran cantidad de códigos binarios posibles.

En este capítulo sólo se han mostrado la suma, la resta y una multiplicación. La implementación del resto de operadores (otras multiplicaciones, división, potencia, etc.) queda algo lejos de este libro, bien por su complejidad, bien porque las técnicas de diseño utilizadas -ASM, RT- están desapareciendo, o bien porque los elementos tecnológicos (PLD, FPGA, VLSI) están fuera del libro. Desde el punto de vista teórico el estudio de estos operadores tiene sentido en libros más enfocados a la estructura de computadores.



CAPÍTULO

7

ELEMENTOS BÁSICOS DE MEMORIA

7.1. Introducción

Al hablar de sistemas combinacionales vimos que en éstos la salida sólo dependía de la entrada, y que su elemento tecnológico principal era la puerta lógica, ya fuera AND, NAND, OR, NOR, etc. La principal característica de un sistema secuencial es que su nueva salida no sólo depende del valor de la entrada, sino también del antiguo valor de la salida, es decir, lo que vaya a hacer el sistema secuencial depende de lo que esté haciendo ahora. Lo anterior quiere decir que el sistema debe tener memoria, debe recordar en qué situación se encuentra. El biestable es el elemento básico de memoria capaz de almacenar un bit.

A continuación se van a presentar las distintas formas que hay de almacenar un bit, y también los principios básicos de este almacenamiento. Este capítulo tiene una gran importancia tanto teórica -por la importancia de los conceptos presentados- como práctica. Si en este capítulo la unidad de información es un único bit y su dispositivo es el biestable, en los siguientes capítulos la unidad de información será la palabra -grupo de bits- y sus dispositivos serán el registro, contador, etc.

7.2. Sincronismo y asincronismo

Un sistema secuencial evoluciona en el tiempo, pasando de un estado a otro según el valor de las entradas. En un sistema secuencial asíncrono cualquier variación en la entrada modifica el estado del sistema. Es decir, un sistema asíncrono *escucha* y procesa todas las variaciones de la entrada.

Sin embargo, en un sistema síncrono el sistema sólo *escucha* y procesa las entradas en unos instantes determinados. Cuando se dice que un sistema está sincronizado quiere decir que está a la *escucha*, y que por tanto procesará las entradas. Y lo contrario, si el sistema no estuviera sincronizado, por mucho que cambiaran las entradas, la salida y el estado del sistema no cambiarían.

Resumiendo, el sincronismo afecta al sistema secuencial en el cuándo, y no en el qué o en el cómo.

Los sistemas secuenciales más fáciles de analizar y diseñar son los síncronos, sin embargo los sistemas más comunes en la *vida real* son los asíncronos, aunque éstos puedan ser convertidos en sistemas síncronos. En el texto se dará más importancia a los sistemas síncronos.

A la hora de entender y aplicar el concepto de sincronismo es fundamental saber cuáles son los momentos de sincronismo. Estos momentos de *escucha* por parte del sistema vienen determinados por una señal llamada Reloj (Clk o Ck del inglés). La señal de reloj es periódica y se define por su:

- Frecuencia. Determina la velocidad a la que evoluciona el reloj, y con él la del sistema secuencial. Se mide generalmente en hercios, kilohercios y megahercios (Hz, KHz y MHz).
- Simetría. Porcentaje de tiempo de un periodo en el que el reloj está a nivel alto o bajo. Generalmente el reloj es simétrico, con un 50% de porcentaje.

En un reloj los hitos que determinan el tipo de sincronismo del sistema son dos:

- Por nivel. *Durante* el nivel activo del reloj el sistema secuencial sigue las variaciones de la entrada. El nivel activo del reloj puede ser el alto o el bajo.
- Por flanco. En el *instante* del flanco activo el sistema evoluciona según el valor de la entrada. El flanco activo puede ser el ascendente (↑) o el descendente (↓) -también denominadas transiciones positivas y negativas: PGT y NGT-.

La diferencia más importante entre sincronismo por nivel o flanco es la duración del sincronismo. Así, por flanco el sistema sólo puede evolucionar una vez (pasar a un único nuevo estado), mientras que por nivel puede hacerlo tantas veces como lo permita la duración del nivel. Los sistemas secuenciales más usados son los síncronos por flanco.

La gráfica 7-1 muestra distintas señales de reloj con sus niveles y flancos.

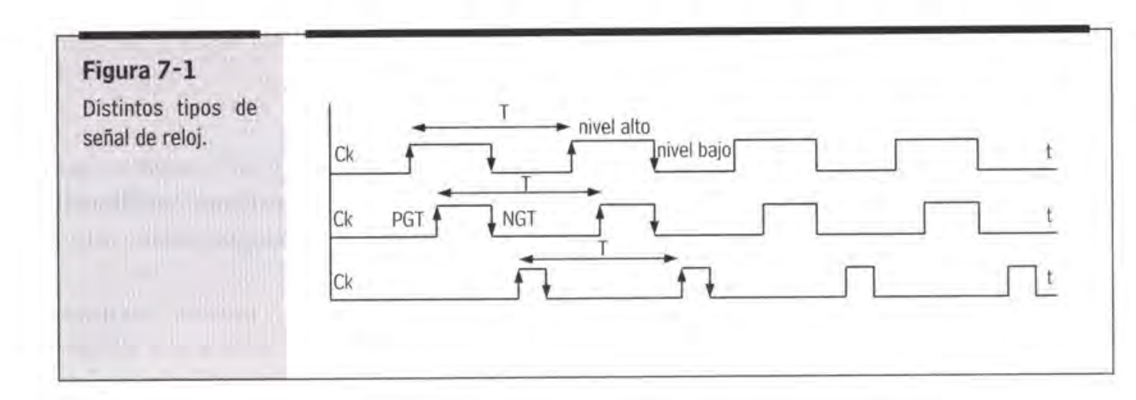


Figura 7-1
Distintos tipos de señal de reloj.

7.3. Técnicas de representación de sistemas secuenciales

La dependencia que del tiempo tienen los sistemas secuenciales conlleva que las técnicas que los representen deban incluir al tiempo como variable. Las técnicas de representación de los sistemas secuenciales, amén de las textuales o gráficas, son las siguientes.

Diagramas de transición de estados (DTE). Un DTE describe de modo gráfico el comportamiento de un sistema. Son de dos tipos -Moore o Mealy- según asocien la salida al estado o a la transición. Las diferencias entre ambos se verán detenidamente en el capítulo 10.

En ambos diagramas los círculos son los estados del sistema y cada arco es una transición entre dos estados para un valor de la entrada. Por ejemplo, el diagrama de la figura 7-2 muestra que en el instante del flanco, del estado Q0 se va al Q2 si la entrada es A=0, o a Q0 si la entrada es A=1.

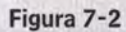
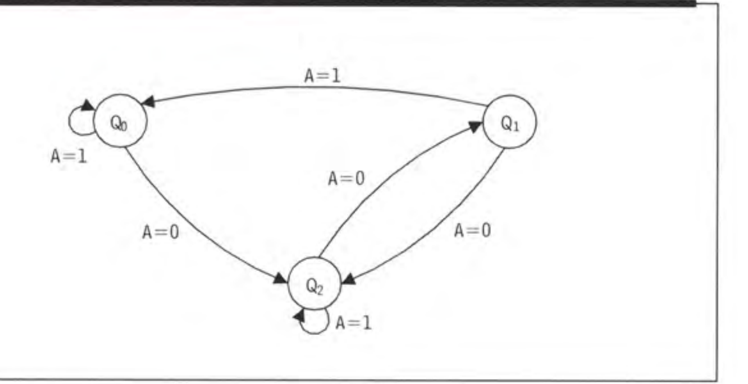


Figura 7-2
Ejemplo de diagrama de transición de estados.



Tablas de Verdad. Genéricamente no cambian respecto de las vistas en capítulos anteriores, pero ahora incluyen dos nuevos conceptos:

- el reloj aparece como señal de entrada y
- hay señales que aparecen en la entrada y en la salida, como t y t+1, respectivamente, o como t-1 y t.

En la tabla 7-1 vemos que si la entrada es AB=00 y el estado anterior era el Q3, al llegar el próximo ↑ el siguiente estado será Q6; y así para el resto de las filas.

Tabla 7-1
Ejemplo de tabla de transición de estados.

Ck	A	B	Qt	Qt+1
↑	0	0	Q3	Q6
↑	0	1	Q2	Q2
↑	1	0	Q4	Q4
↑	1	1	Q0	Q1
--	--	--	--	--
--	--	--	--	--

Cronogramas. Esta representación gráfica incluye el tiempo como variable. Así vemos cómo en la figura 7-3 en t_1 la salida es cero y en t_2 la salida es 1, siendo A y B iguales en ambos casos.

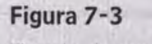
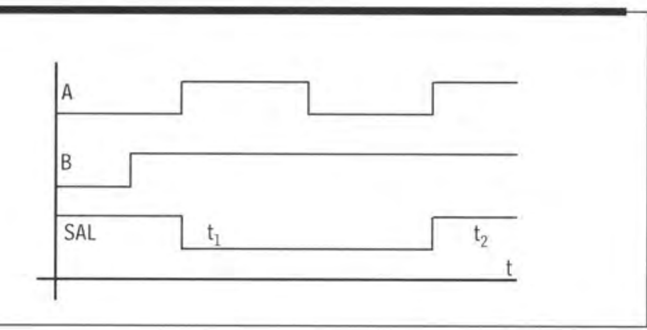


Figura 7-3
Ejemplo de cronograma.



7.3.1. Biestables

Recordando lo dicho en la introducción, un sistema secuencial es aquel que es capaz de conocer el estado en el que se encuentra, es decir, un sistema secuencial tiene memoria.

Un biestable es el dispositivo encargado de almacenar un bit -ya sea 1 o 0-, y de mantener dicho valor hasta que sea sustituido por otro. Es decir, en un biestable, una vez desaparecida la acción que hizo almacenar el 1 o el 0, su efecto no desaparecerá.

Un biestable puede ser de diferentes tipos según su lógica de disparo y según su sincronismo.

- *Sincronismo*. En un biestable asíncrono todas las variaciones de la entrada pueden afectar a la salida, mientras que a un biestable síncrono sólo le afectan las variaciones de la entrada producidas durante el nivel o flanco activo del reloj. Un biestable puede ser síncrono por nivel, flanco, maestro-esclavo y maestro-esclavo con cierre de datos.
- *Lógica de disparo*. Cada tipo de biestable tiene distintas entradas y evoluciona ante ellas de modo distinto, aunque en todos los casos mantenga la función fundamental de almacenamiento. O sea, cada lógica de disparo hace evolucionar de distinto modo al biestable. Los biestables pueden ser de tipo R-S, D, J-K, T, etc.

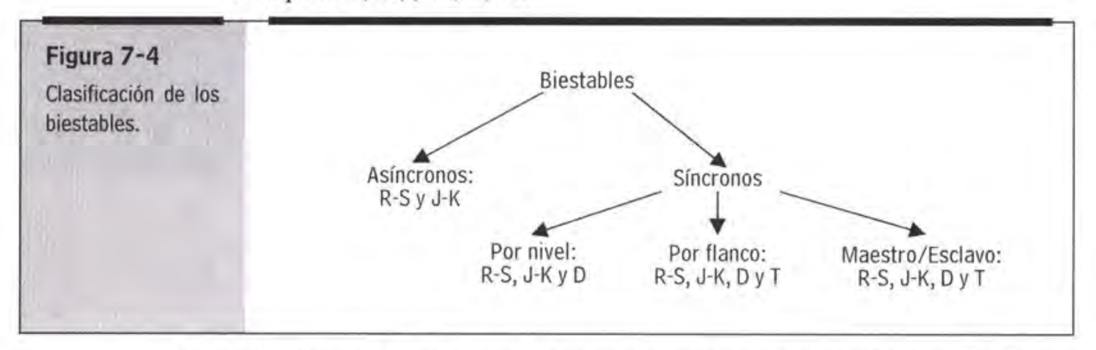


Figura 7-4
Clasificación de los biestables.

En la clasificación de la figura 7-4 vemos cómo determinadas lógicas de disparo son válidas para cualquier sincronismo, y cómo otras sólo lo son para flanco o Maestro-Esclavo. En la tabla 7-2 los asteriscos (*) indican que dichos biestables son de uso común y están implementados en CI de tipo MSI, y las X indican que el biestable en cuestión tiene interés teórico pero no práctico.

Tabla 7-2
Tipos básicos de biestables.

	R-S	J-K	D	T
Asíncrono	*	X		
Por nivel	X	X	*	
Por flanco	X	*	*	X
Maestro-Esclavo	*	*	*	X

Antes de comenzar con el desarrollo, merece la pena dedicar un instante a la nomenclatura. En el texto utilizaremos preferentemente el término biestable, pero también existen latch y flip-flop. El primer término significa cerrojo -para sincronía por nivel-, mientras el segundo es un ejemplo claro de terminología anglosajona.

En los siguientes apartados describiremos los distintos biestables, y destacaremos para cada uno de ellos su:

- Circuito lógico.
- Descripción del sincronismo.
- Tabla de funcionamiento y tabla de excitación.
- Tabla de verdad y ecuaciones booleanas.
- Diagrama de transición de estados.
- Cronograma aclaratorio.
- Situaciones anómalas y peligrosas.

Primeramente describiremos todos los biestables asíncronos, luego los síncronos por nivel y por último los síncronos por flanco y tipo Maestro/Esclavo.

7.4. Biestables asíncronos

En un biestable asíncrono no hay reloj que sincronice su evolución y así la salida evoluciona frente a todos los cambios de la entrada, es decir: todo cambio en la entrada conllevará un posible cambio en la salida.

Desde el punto de vista descriptivo los biestables asíncronos preceden a los síncronos, y en ambos el concepto que permite la posibilidad de memoria es la realimentación de la salida a la entrada, o dicho de modo más secuencial: la nueva salida o estado (Q_{t+1}) depende de la entrada (E_t) y de la salida o estado actual (Q_t), $Q_{t+1}=f(E_t, Q_t)$.

La figura 7-5 es el ejemplo más sencillo de realimentación. Como vemos en el cronograma, si la salida inicialmente es cero y la entrada no cambiara, la salida seguiría a cero. Ahora bien, si la entrada pasara a 1 la salida también lo haría, puesto que $S = E+S = 1+S = 1$. Si en este momento la entrada pasara a 0-E =0, vemos que la salida memorizaría su anterior valor puesto que $S = E+S = E+1 = 1$, es decir, desaparecida la acción generatriz (E=1) se mantiene la acción generada (S=1). El problema de este primer biestable es que una vez puesto a 1 no puede volver a 0, lo que le hace inservible en la práctica.

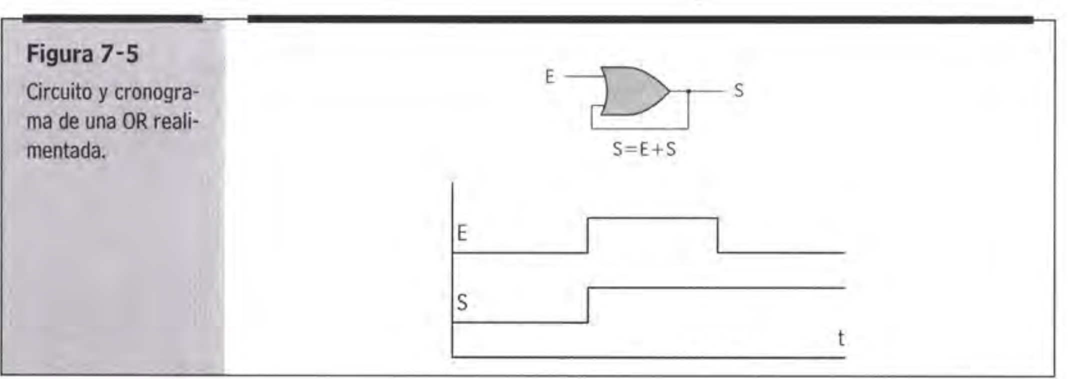


Figura 7-5
Circuito y cronograma de una OR realimentada.

Este sencillo circuito nos ha permitido establecer la relación entre memoria y realimentación, nos queda por tanto *ajustar* esa realimentación para que sea útil. En los siguientes apartados veremos los biestables R-S y J-K que solventan el problema anterior.

7.4.1. Biestable R-S asíncrono

Un biestable R-S asíncrono consta de dos entradas R (reset) y S (set) y de dos salidas Q y $\overline{Q}$. El biestable se implementa con dos puertas NOR o NAND con sus salidas realimentadas. En el primer caso las entradas R y S son activas por nivel alto, mientras que en el segundo las entradas resultan activas por nivel bajo.

7.4.1.1. Biestable R-S asíncrono con puertas NOR

El circuito que implementa un biestable R-S con puertas NOR es el de la figura 7-6. Su comportamiento viene descrito por la tabla de verdad 7-3.

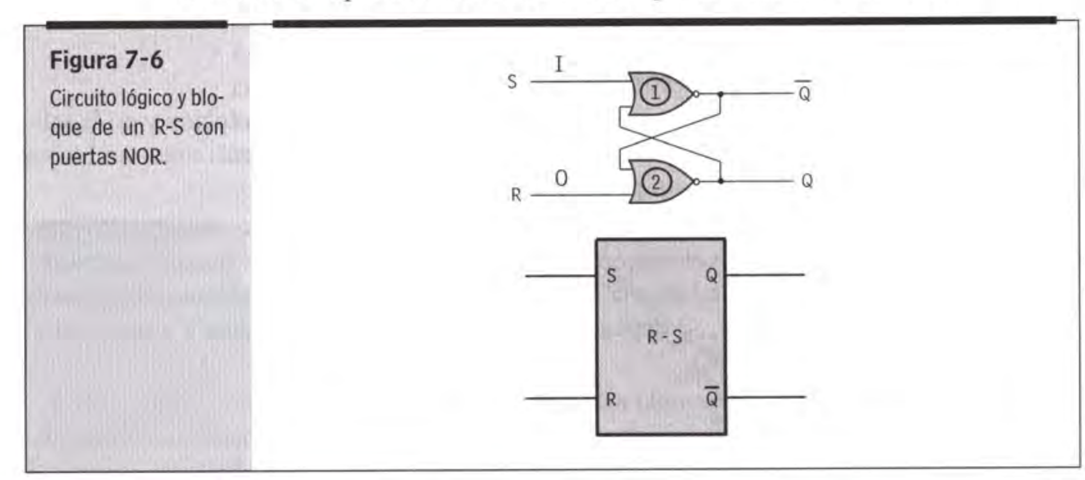


Figura 7-6

Circuito lógico y bloque de un R-S con puertas NOR.

Tabla 7-3

Tabla de un R-S con puertas NOR.

S	R	Q	$\overline{Q}$	Operación
1	0	1	0	Puesta a 1
0	1	0	1	Puesta a 0
0	0	Q_{t-1}	$\overline{Q_{t-1}}$	Memoria
1	1	0	0	Prohibido

Comprobemos la tabla de verdad según las ecuaciones de las salidas:

$$Q = \overline{R + \overline{Q}} \qquad \overline{Q} = \overline{S + Q}$$

1. Si S=1 y R=0. PUESTA A 1.

$$\overline{Q} = \overline{S + Q} = 0 \quad \text{y} \quad Q = \overline{R + \overline{Q}} = 1$$

$$Q=1 \quad \text{y} \quad \overline{Q} = 0$$

2. Si S=0 y R=1. PUESTA A 0

$$Q = \overline{R + \overline{Q}} = 0 \quad \text{y} \quad \overline{Q} = \overline{S + Q} = 1;$$

$$Q = 0 \quad \text{y} \quad \overline{Q} = 1$$

3. Si S=0 y R= 0. MANTENIMIENTO (introducimos el tiempo en las ecuaciones)

$$Q_t = \overline{R + \overline{Q_{t-1}}} = Q_{t-1}$$

$$\overline{Q_t} = \overline{S + Q_{t-1}} = \overline{Q_{t-1}};$$

$$Q_t = Q_{t-1} \quad \text{y} \quad \overline{Qt} = \overline{Q_{t-1}}$$

Si la báscula estaba a 1 sigue a 1, y si estaba a 0 sigue a 0.

4. Si S=1 y R=1. PROHIBIDO

$$Q = \overline{R + \overline{Q}} = 0 \quad \overline{Q} = \overline{S + Q} = 0;$$

$$Q = 0 \quad \text{y} \quad \overline{Q} = 0$$

En este caso tanto Q como $\overline{Q}$ son 0, violando la condición de contrarias de su nombre. El biestable queda en un estado definido pero ilógico o absurdo, por eso decimos que S=R=1 es una situación prohibida o no deseada. De hecho, si activamos S=1 y R=1 estamos dando la orden simultánea de poner a 1 y 0 el biestable, lo que no parece muy normal.

A modo de resumen vemos que el biestable R-S descrito puede ponerse a 1 (S=1 y R=0), puede ponerse a 0 (S=0 y R=1) o puede quedarse en reposo (S=0 y R=0), y que además se debe evitar la situación prohibida (S=1 y R=1), cuya salida es conocida y estable, pero indeseable.

7.4.1.2. Biestable R-S asíncrono con puertas NAND

Teóricamente, este biestable es idéntico al implementado con puertas NOR, sólo se distingue de áquel en que sus entradas son activas por nivel bajo. Este biestable con puertas NAND es más utilizado que el anterior (figura 7-7).

Figura 7-7

Circuito lógico y bloque de un R-S con puertas NAND.

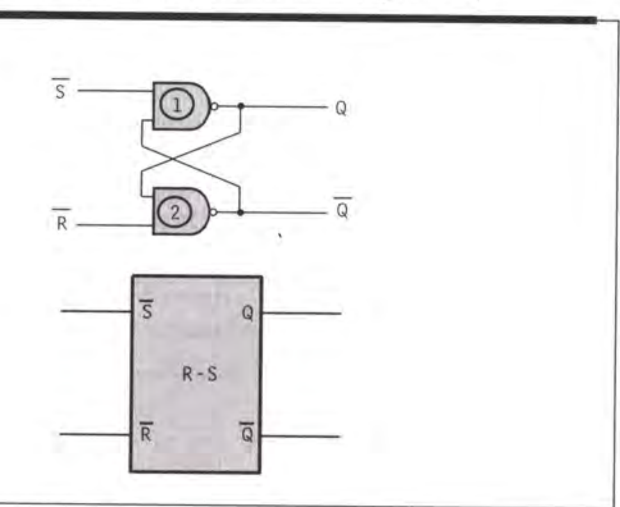


Tabla 7-4

Tabla de R-S con puertas NAND.

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	Operación
0	1	1	0	Puesta a 1
1	0	0	1	Puesta a 0
1	1	Q_{t-1}	$\overline{Q_{t-1}}$	Mantenimiento
0	0	0	0	Prohibido

Comprobemos la tabla de verdad según las ecuaciones de salida:

$$Q = \overline{\overline{S} \cdot \overline{Q}} \qquad \overline{Q} = \overline{\overline{R} \cdot Q}$$

1. Si $\overline{S} = 0$ y $\overline{R} = 1$. PUESTA A 1

$$Q = \overline{\overline{S} \cdot \overline{Q}} = 1 \quad \text{y} \quad \overline{Q} = \overline{\overline{R} \cdot Q} = 0$$

resulta $Q = 1$ y $\overline{Q} = 0$

2. Si $\overline{S} = 1$ y $\overline{R} = 0$. PUESTA A 0

$$Q = \overline{\overline{S} \cdot \overline{Q}} = 0 \quad \text{y} \quad \overline{Q} = \overline{\overline{R} \cdot Q} = 0$$

resulta $Q = 0$ y $\overline{Q} = 1$

3. Si $\overline{S} = 1$ y $\overline{R} = 1$. MANTENIMIENTO

$$\overline{Q_t} = \overline{\overline{R} \cdot Q_{t-1}} = \overline{1 \cdot Q_{t-1}} = \overline{Q_{t-1}}$$

$$Q_t = \overline{\overline{S} \cdot \overline{Q_{t-1}}} = \overline{1 \cdot \overline{Q_{t-1}}} = Q_{t-1};$$

resulta $Q_t = Q_{t-1}$ y $\overline{Q_t} = \overline{Q_{t-1}}$

Si la báscula estaba a 1 sigue a 1, y si estaba a 0 sigue a 0

4. Si $\overline{S} = 0$ y $\overline{R} = 0$. PROHIBIDO

$$Q = \overline{\overline{S} \cdot \overline{Q}} = 1 \qquad \overline{Q} = \overline{\overline{R} \cdot Q} = 1$$

resulta $Q = 1$ y $\overline{Q} = 1$

La situación se corresponde con la encontrada para las puertas NOR. Si la entrada es tal que activamos la puesta a 1 y a 0 la salida es ilógica; en este caso ambas salidas a 1.

El cronograma de la figura 7-8 describe la evolución del R-S asíncrono con puertas NAND, partiendo de un valor incial 0 en la báscula (Q=0).

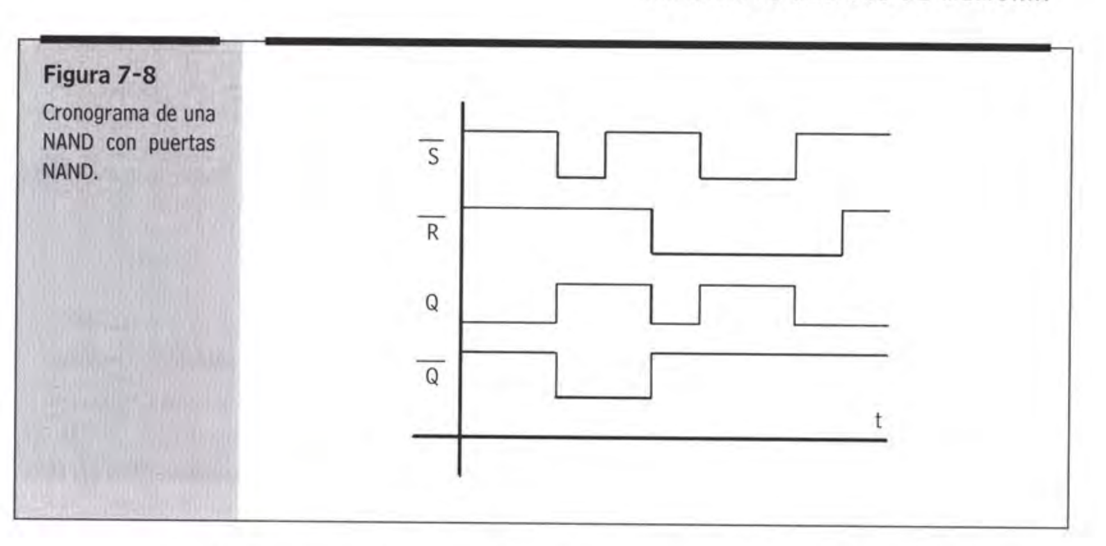


Figura 7-8
Cronograma de una NAND con puertas NAND.

En el cronograma de la figura 7-8 vemos cómo cada vez que $\overline{S} = 0$ o $\overline{R} = 0$ el biestable se pone a 1 o a 0, respectivamente. Si $\overline{S} = \overline{R} = 0$ ambas salidas valen 1 quedando la báscula en un estado ilógico. Este estado ilógico desaparecerá en cuanto los estados tomen un nuevo valor lógico, excepto mantenimiento ($\overline{S} = \overline{R} = 1$).

7.4.1.3. Retardos en un biestable R-S

En el cronograma de la figura 7-9 están explicitados los retardos asociados a cada puerta NAND. Vemos que en la puesta a 1 es Q quien arrastra a $\overline{Q}$ (la puerta superior a la inferior), y que en la puesta a 0 es al revés. Este orden es observable en las anteriores ecuaciones booleanas.

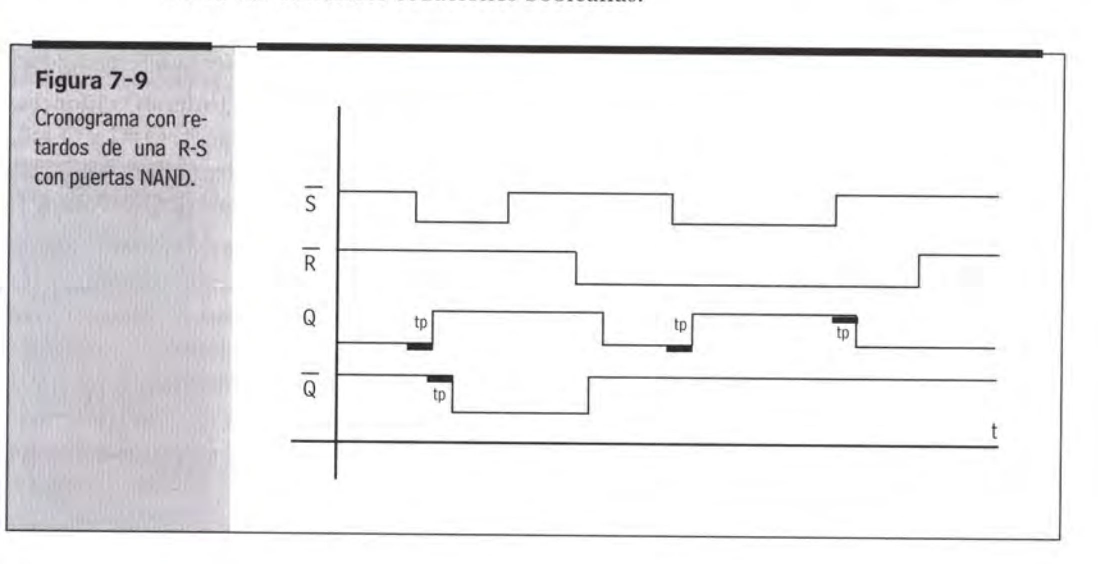


Figura 7-9
Cronograma con retardos de una R-S con puertas NAND.

Cada retardo marcado es tp (tiempo de puerta) y así el tiempo de evolución de un biestable asíncrono, ya sea con puertas NAND o NOR, es

tbiestable = 2 x tpuerta

es decir, desde que la entrada cambia pasan 2 x tp segundos hasta que su efecto es estable en la salida (ver el apartado 7.12 de parámetros de básculas).

7.4.1.4. Situación prohibida y sus consecuencias

Como hemos visto, si se activaban ambas entradas las dos salidas pasaban a 1. Esta situación es tan absurda como estable, aunque todavía puede empeorar.

Veamos, en una báscula R-S NAND, si estando ambas salidas a 1 pasamos a puesta a 1 o a 0, la salida pasa a un nuevo valor estable y lógico, pero si de $\overline{S} = \overline{R} = 0$ pasamos a $\overline{S} = \overline{R} = 1$ la salida quedará indeterminada; tomará un valor, sí, pero ese valor será a priori desconocido para nosotros.

Observemos la situación desde el álgebra de Boole:

Si $\overline{S} = \overline{R} = 0$ implica $Q = \overline{Q} = 1$ y si seguidamente $\overline{S} = \overline{R} = 1$, resulta que:

$$Q = \overline{\overline{S} \cdot \overline{Q}} = \overline{1 \cdot 1} = 0 \text{ y luego}$$

$$\overline{Q} = \overline{\overline{R} \cdot Q} = \overline{1 \cdot 1} = 0 \text{ y luego}$$

$$Q = \overline{\overline{S} \cdot \overline{Q}} = \overline{1 \cdot 0} = 1 \text{ y luego}$$

$$\overline{Q} = \overline{\overline{R} \cdot Q} = \overline{1 \cdot 0} = 1\text{, y luego...}$$

Si ambas puertas son igual de rápidas, la situación deriva en un continuo cambio de 0 a 1 y de 1 a 0, quedando la báscula en una situación denominada *carreras*. Sin embargo, por fabricación, ambas puertas nunca pueden ser igual de rápidas.

- *Si la puerta 1* (superior) es más rápida que la puerta 2 (inferior) entonces: Q fija antes su valor, y arrastra a $\overline{Q}$, quedando la báscula a 1: $Q=1$ y $\overline{Q} = 0$.
- *Si la puerta 2* (inferior) es más rápida que la puerta 1 (superior) entonces: $\overline{Q}$ fija antes su valor, y arrastra a Q, quedando la báscula a 0: $Q=0$ y $\overline{Q} = 1$.

Conclusión: si las entradas $\overline{S}$ y $\overline{R}$ pasan de activas a inactivas simultáneamente (de 00 a 11) la báscula quedará a 0 o a 1, según qué puerta sea más rápida; pero como no es posible saber qué puerta es más rápida, no sabemos a priori en qué estado quedará la báscula. Es decir, no sabemos qué está haciendo nuestro sistema. Hemos visto que de una situación absurda pero definida, hemos pasado a una indefinida. También hemos visto que si bien el causante de la indeterminación ha sido el paso a $\overline{S} = \overline{R} = 1$, en realidad la culpa no es de esta situación, sino de la anterior: $\overline{S} = \overline{R} = 0$.

La figura 7-10 muestra tres cronogramas. En el primero es la puerta 2 la más rápida, en el segundo es la 1 la más rápida y en el tercero ambas son igual de rápidas -caso teórico-.

Figura 7-10 (a)

Cronograma de un R-S asíncrono con NAND.

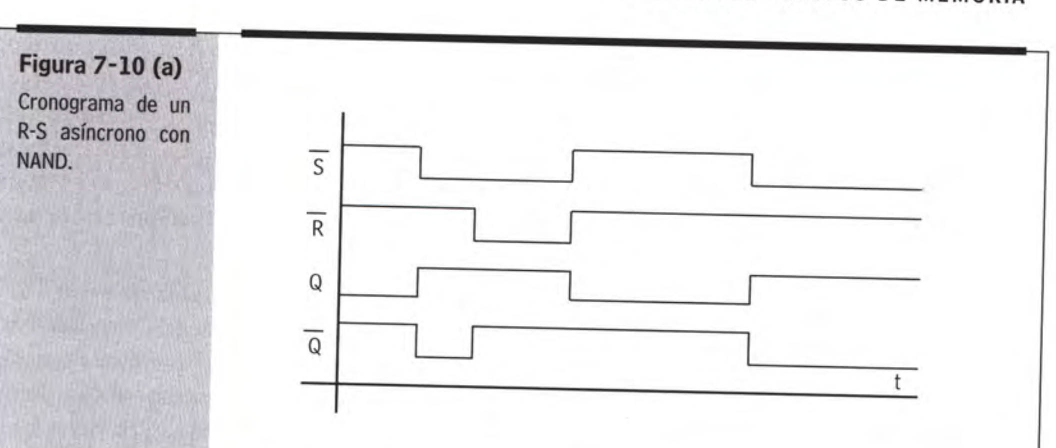


Figura 7-10 (b)

Cronograma de un R-S asíncrono con NAND.

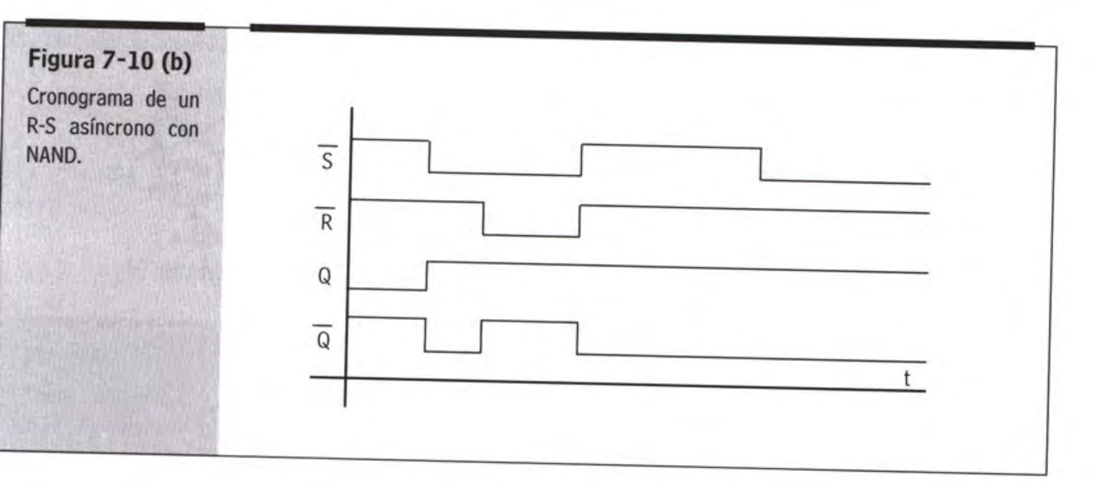
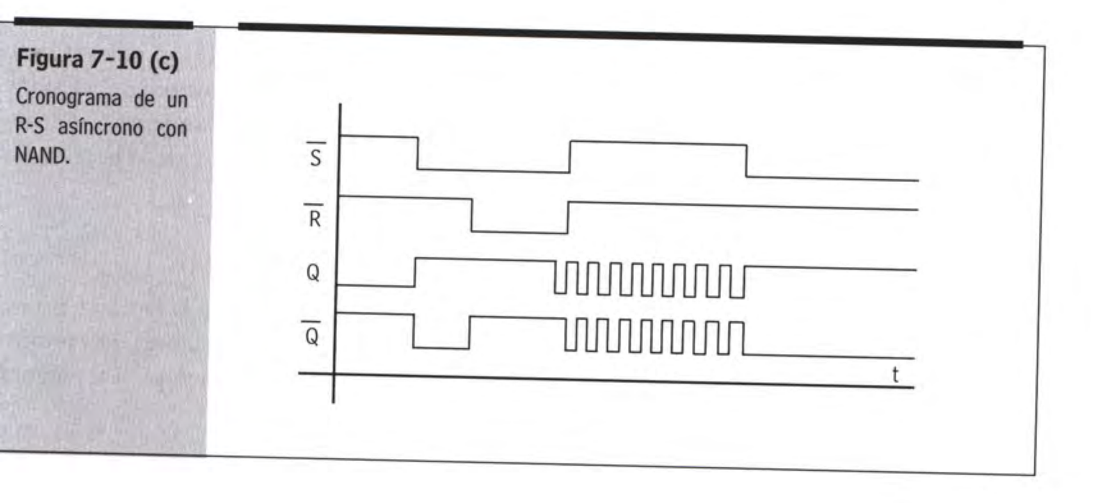


Figura 7-10 (c)

Cronograma de un R-S asíncrono con NAND.



Lo visto en los cronogramas anteriores y ecuaciones es cierto desde un punto de vista teórico, pero no lo es en la práctica. Al igual que hemos dicho que nunca pueden ser iguales las dos puertas, también podemos afirmar que es imposible desactivar las dos entradas simultáneamente, ya que siempre cambiará antes una de las dos, y ésta arrastrará a la otra. Así pues, el problema descrito es de un gran valor teórico, pero con poco interés práctico. Esta situación cambiará a peor con el sincronismo.

La situación de indeterminación no es contemplada por la tabla de verdad 7.4, ya que ésta no considera transiciones, sino valores estáticos de las entradas. Esta incompletitud no afecta al uso de la tabla de verdad, puesto que esta situación no es ni deseada ni común. Ahora bien, si quisiéramos representar el comportamiento global del biestable R-S asíncrono con entradas activas por nivel bajo, recurriríamos al diagrama de transición de estados de la figura 7-11.

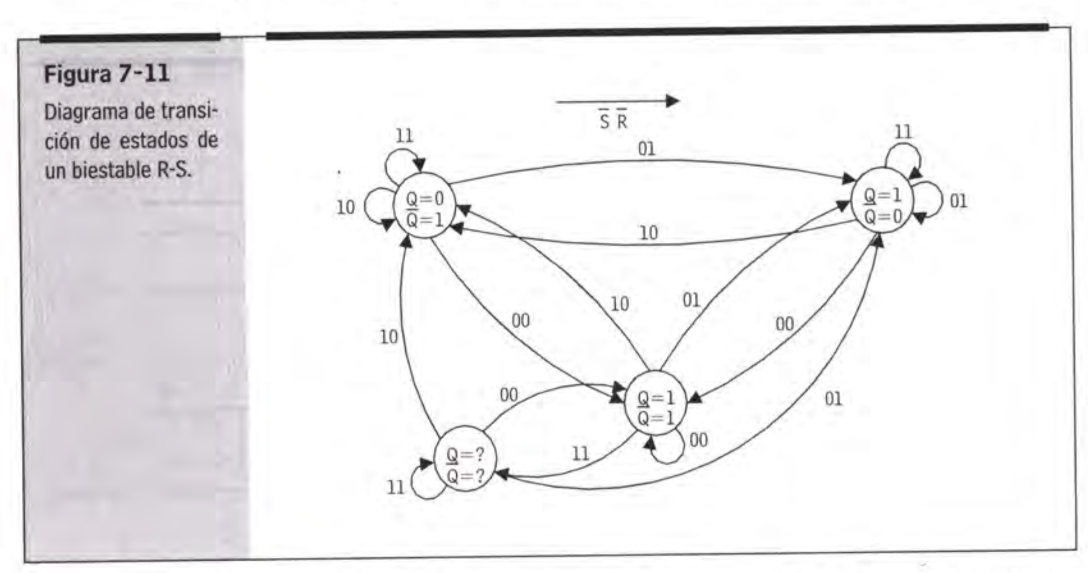


Figura 7-11
Diagrama de transición de estados de un biestable R-S.

En el anterior diagrama, la parte superior, formada por los dos estados $Q=0$ y $\overline{Q}=1$ y $Q=1$ y $\overline{Q}=0$, son los estados del funcionamiento normal del biestable, mientras que el estado inferior izquierda describe el comportamiento anómalo o prohibido. Queda para el lector el repaso del comportamiento de la báscula R-S a la vista del DTE de la figura 7-11.

7.4.1.5. Ecuación y uso del biestable R-S

Anteriormente hemos visto cómo evoluciona la salida frente a la entrada. En este apartado plantearemos el análisis en sentido inverso: ¿qué valores deben tomar las entradas para pasar a un estado deseado, partiendo del actual? La pregunta anterior se responde en la tabla de verdad 7-5.

BIBLIOTECA CADIZ PÚBLICA

Tabla 7-5

Bloque y tabla de un biestable R-S.

Q_t	Q_{t+1}	$\overline{S}$	$\overline{R}$
0	0	1	X
0	1	0	1
1	0	1	0
1	1	X	1

Es decir, si el biestable tiene ahora un 0 y el siguiente valor debería ser un 0, las entradas deben ser $\overline{S}=1$ y $\overline{R}=X$ (cualquier valor). Y así sucesivamente para los otros casos. La tabla 7-5 es utilizada en diseño de sistemas secuenciales, objetivo de los siguientes capítulos.

La tabla 7-6 describe el comportamiento de la báscula R-S, y de ella podemos obtener y simplificar el correspondiente diagrama de V-K.

Tabla 7-6

Tabla de verdad, V-K e implementación de un R-S.

$\overline{S}$	$\overline{R}$	Q_t	Q_{t+1}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$Q_{t+1} = S + Q_t \bullet \overline{R}$

El circuito resultante (7-6) puede sustituir al original con dos NAND, y de hecho así suele ser en diseños de sistemas secuenciales asíncronos.

7.4.1.6. Resumen del R-S asíncrono

En el biestable R-S asíncrono las salidas evolucionan frente a todos los cambios en la entrada según la tabla de verdad 7-4. La activación de S (ya sea por nivel activo bajo o alto) pone el biestable a 1, la activación de R lo pone a 0 y la no activación ni de R ni de S hace que el biestable memorice la última salida generada. Por último, no se deben activar simultáneamente R y S, pues la situación derivada puede ser indeterminada.

7.4.2. Biestable J-K asíncrono

En este caso, el biestable J-K modifica al R-S de modo que se resuelven los problemas detectados en el R-S, o por lo menos nos acercamos a su resolución.

En este caso las entradas son dos: J y K, ambas activas por nivel alto. La entrada J se corresponde con la S del R-S, y la K con la R. El carácter asíncrono del J-K hace que sus salidas varíen frente a cualquier evolución de la entrada.

Si analizamos el circuito de la figura 7-12 obtendremos la tabla de verdad 7-7 del biestable J-K asíncrono. Como vemos, el circuito tiene una doble realimentación y su parte interna se corresponde con un R-S por nivel bajo, lo que será tenido en cuenta para obtener la tabla de verdad 7-7.

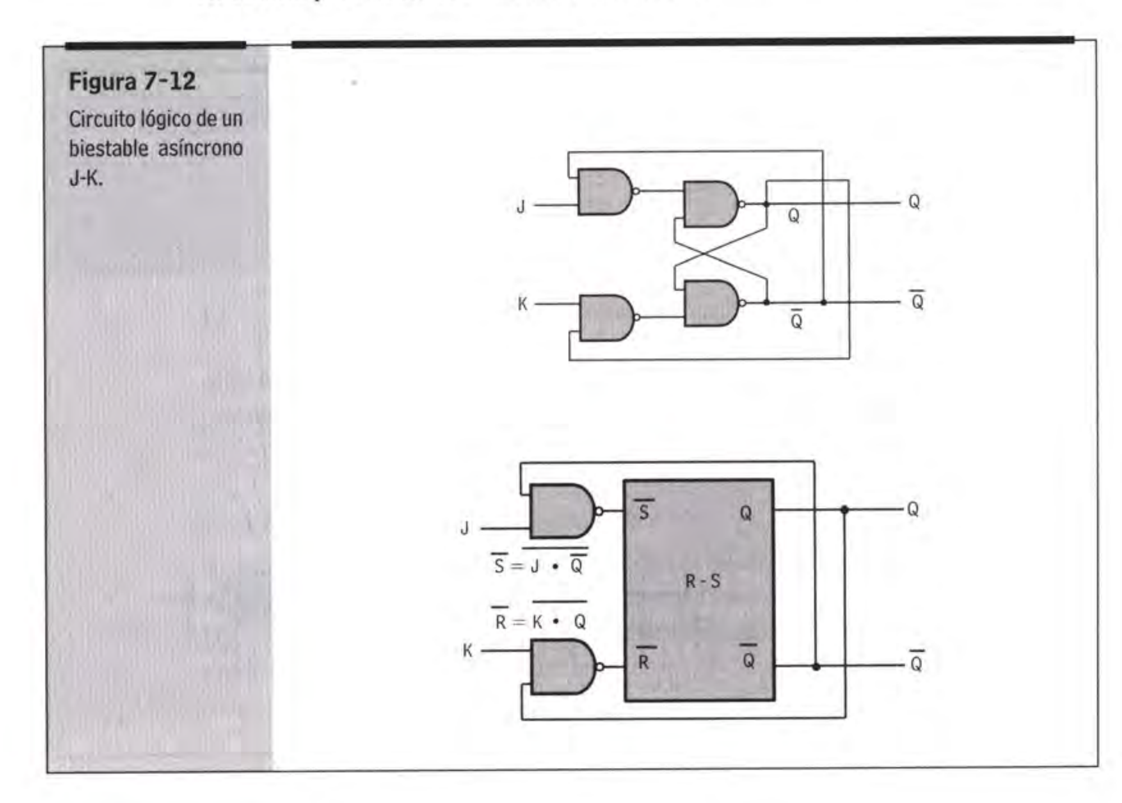


Figura 7-12
Circuito lógico de un biestable asíncrono J-K.

Tabla 7-7
Bloque y tabla de un biestable J-K.

J	K	Q	$\overline{Q}$	Operación
1	0	1	0	Puesta a 1
0	1	0	1	Puesta a 0
0	0	Q_{t-1}	$\overline{Q_{t-1}}$	Memoria
1	1	Carreras		Prohibido

Analicemos el circuito desde el álgebra de Boole:

1. J=1 y K=0. PUESTA A 1

$$\overline{S} = \overline{J \cdot \overline{Q}} = Q_{t-1} \quad y \quad \overline{R} = \overline{K \cdot Q} = 1;$$

Si $\overline{S} = Q_{t-1}$ y $\overline{R} = 1$ entonces $Q_t = 1$ y $\overline{Q_t} = 0$

2. J=0 y K=1. PUESTA A 0

$$\overline{S} = \overline{J \cdot \overline{Q}} = 1 \quad y \quad \overline{R} = \overline{K \cdot Q} = \overline{Q_{t-1}}:$$

Si $\overline{S} = 1$ y $\overline{R} = \overline{Q_{t-1}}$ entonces: $Q_t = 0$ y $\overline{Q_t} = 1$

3. J=0 y K=0. MANTENIMIENTO

$$\overline{S} = \overline{J \cdot \overline{Q}} = 1 \quad y \quad \overline{R} = \overline{K \cdot Q} = 1;$$

Si $\overline{S} = 1$ y $\overline{R} = 1$ entonces: $Q_t = Q_{t-1}$ y $\overline{Q_t} = \overline{Q_{t-1}}$

4. J=1 y K=1. CARRERAS. PROHIBIDO

$$\overline{S} = \overline{J \cdot \overline{Q}} = Q_{t-1} \quad y \quad \overline{R} = \overline{K \cdot Q} = \overline{Q_{t-1}}$$

$$\overline{S} = Q_{t-1} \quad y \quad \overline{R} = \overline{Q_{t-1}}$$

En este último caso, si partiéramos arbitrariamente de $Q_{t-1} = 0$ y $\overline{Q_{t-1}} = 1$ entonces $\overline{S} = 0$ y $\overline{R} = 1$ y por tanto $Q_t = 1$ y $\overline{Q_t} = 0$, lo que conllevaría un nuevo salto si J y K siguieran a 1. Esto a su vez generaría nuevos valores para Q y $\overline{Q}$, y así sucesivamente. Las dos salidas del biestable J-K estarán basculando de 0 a 1 sin parar. A este fenómeno se le denomina *carreras*. Este comportamiento, aunque lógico, es totalmente inestable e indeseable.

El comportamiento anterior se observa claramente en el cronograma de la figura 7-13.

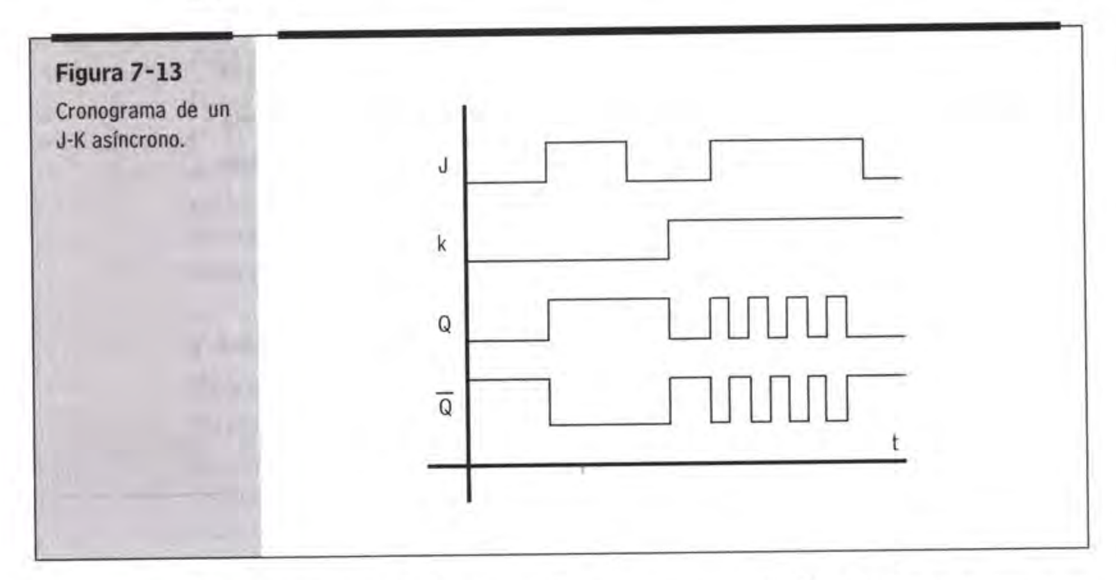


Figura 7-13
Cronograma de un J-K asíncrono.

En el cronograma 7-13 vemos cómo se comporta el J-K, y cómo se representan las correspondientes carreras. También muestra cómo después de las carreras el biestable puede volver a un comportamiento normal, siempre que las entradas JK pasen a 01 o 10.

Si nos fijamos en el tiempo de evolución del biestable vemos que éste consume el tiempo asociado al R-S y el tiempo de las dos primeras NAND que actúan en paralelo, así:

tbiestable = tp + 2 x tp = 3 x tp

7.4.2.1. Situaciones prohibidas y sus consecuencias

Mientras las entradas del biestable se encuentren activas (J=K=1), las salidas estarán basculando a gran velocidad. Si por ejemplo dichas salidas activaran a elementos electromecánicos, éstos estarían conectándose y desconectándose de forma continua, con el correspondiente perjucio para el sistema. Como ya hemos dicho, dicha situación desaparecería si J y K pasaran a 10 o 01, quedando la salida a 1 y a 0, respectivamente. Pero si durante la situación de carreras, J y K se desactivaran simultáneamente (J=K=0), entonces el biestable pasaría a memorizar o mantener el último valor alcanzado, pero éste no es conocido a priori, ya que el basculamiento es muy rápido (del orden de MHz).

El cronograma de la figura 7-14 muestra lo enunciado, y en él vemos cómo después de una situación inestable (carreras) pasamos a una situación indeterminada. Finalmente la salida deja de ser desconocida, pues las entradas pasan a JK = 01.

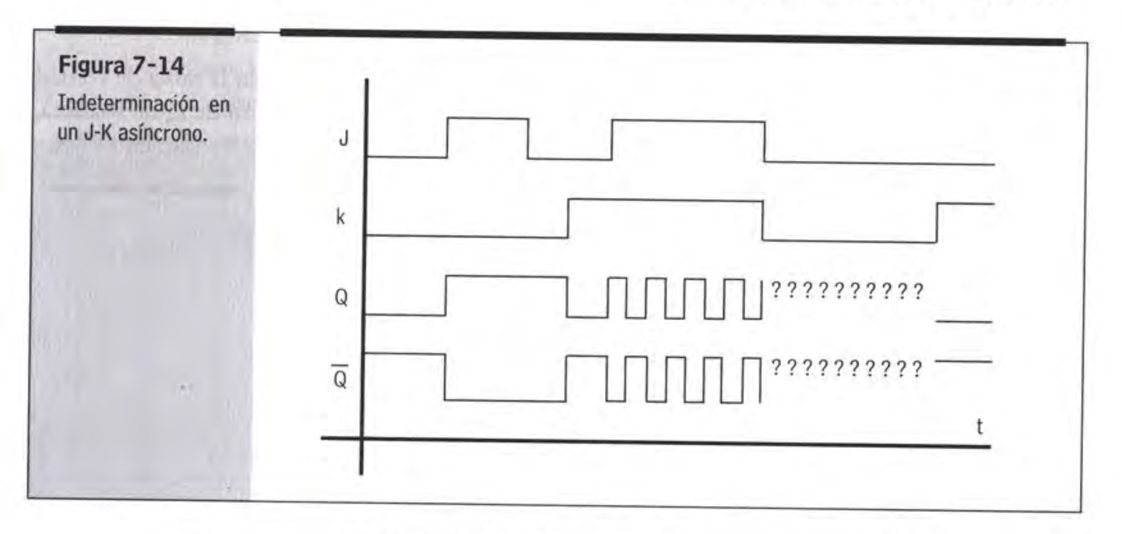


Figura 7-14
Indeterminación en un J-K asíncrono.

Como vemos, la situación de indeterminación se da tanto en el R-S como en el J-K, y ambos en el mismo supuesto: que las entradas pasen de estar activas a inactivas. Pero ambas situaciones, la del R-S y la del J-K, son distintas:

- En el J-K la situación prohibida es lógica pero inestable, mientras que en el R-S es ilógica pero estable.
- La indeterminación en el R-S se resuelve para cada biestable en particular, no así para el J-K.
- La situación prohibida es peor en el J-K que en el R-S. El basculamiento continuo del J-K puede producir malos funcionamientos en otros dispositivos.

De esta comparación se deduce que es peor el biestable J-K asíncrono que el R-S, pero esto no será así cuando cuando introduzcamos el sincronismo en el diseño de los biestables.

El diagrama de transición de estados de la figura 7-15 muestra al completo el comportamiento de un biestable J-K asíncrono. De nuevo resulta interesante que el lector repase el J-K a la luz del DTE.

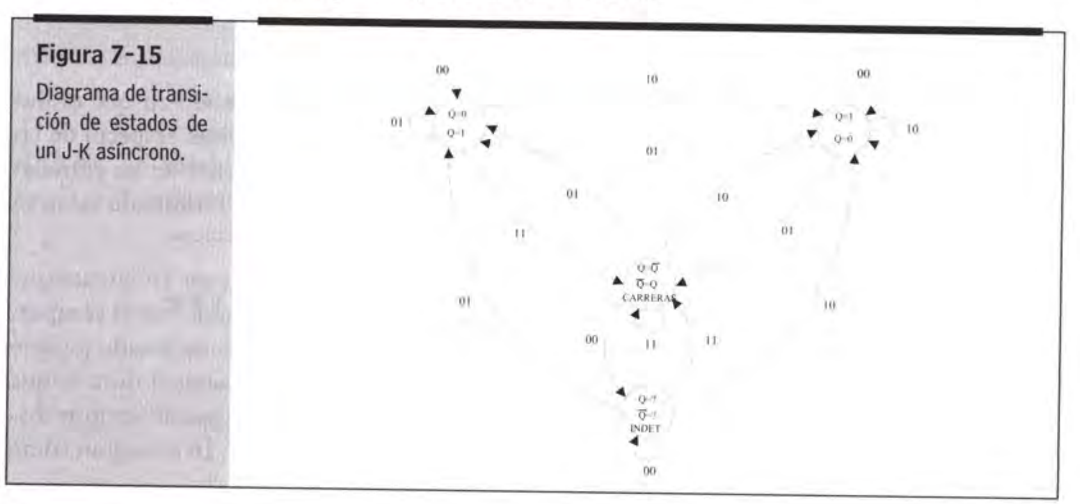


Figura 7-15
Diagrama de transición de estados de un J-K asíncrono.

7.4.2.2. Uso del biestable J-K

Al igual que con el R-S, veamos cómo usar el biestable J-K. En la tabla de verdad 7-8 vemos que en realidad ambos biestables asíncronos se usan de igual manera, sólo cambia el nivel de actividad de las entradas, bajo en el R-S y alto en el J-K.

Tabla 7-8

Bloque y tabla de un biestable R-S.

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

7.4.2.3. Resumen del J-K asíncrono

Podemos decir lo mismo que para el R-S asíncrono: la activación de J hace que Q se ponga a 1 (Q=1), la activación de K pone la salida a 0 (Q=0), la no activación ni de J ni de K hace que se mantenga en la salida su último valor, y, por último, nunca deben activarse ambas entradas a la vez, pues se producirían carreras.

Aunque parezcan iguales el R-S y el J-K, no lo son, como bien muestra su circuito y el análisis de la situación prohibida, y como confirmará la evolución de ambos biestables frente al sincronismo.

7.5. Biestables síncronos por nivel

Como se dijo en el punto 7.3, el sincronismo no afecta al modo en que evoluciona un biestable, sólo afecta al *cuándo* evolucionan las salidas respecto de las entradas. Así, en un biestable síncrono por nivel las variaciones de las entradas sólo afectan a la salida si se producen cuando el reloj toma determinado valor, ya sea el 0, el 1, el flanco ascendente o el descendente.

Al hablar de un sistema secuencial con memoria hay que tener en cuenta que una pequeña variación de la entrada en el tiempo, puede modificar el comportamiento del biestable durante mucho tiempo memorizando un estado espurio pasajero. Por ejemplo, el efecto de un glitch en un combinacional dura lo que dura éste, pero si este glitch afecta a un biestable el resultado puede ser muy distinto del esperado. El circuito y el cronograma de la figura 7-16 muestran cómo

el glitch producido por un inversor al pasar de A=B=C=0 a A=C=1 y B=0 tiene un efecto perdurable en el biestable. Frente a esta situación el sincronismo es muy importante: aporta orden y seguridad.

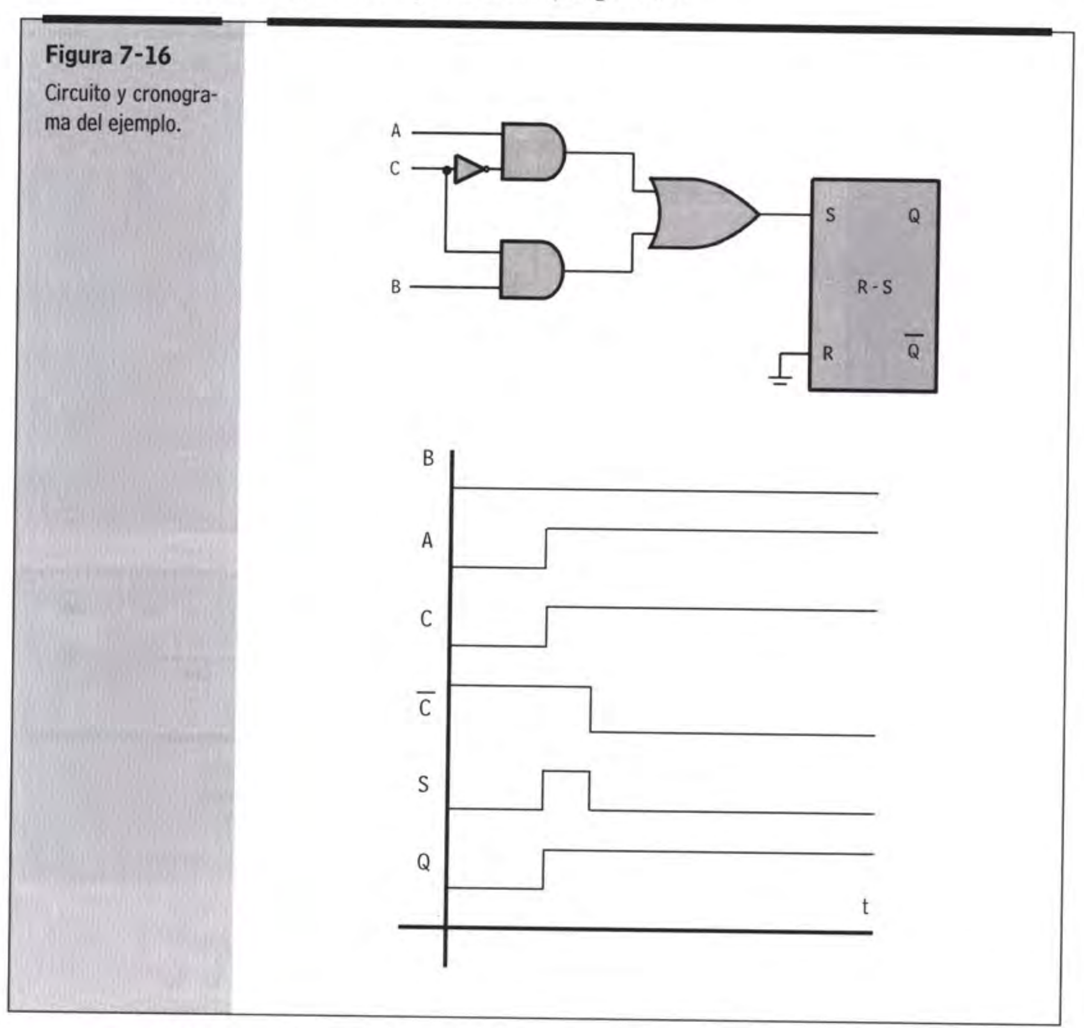


Figura 7-16
Circuito y cronograma del ejemplo.

El sincronismo puede evitar la anterior situación si conseguimos ordenar los eventos para sincronizarlos, así, primero evolucionarán A, B y C, y cuando estén estabilizados el reloj tomará un valor activo para que el biestable opere con entradas libres de glitches. El sincronismo siempre es beneficioso y deseable para un circuito.

7.5.1. Biestables R-S y J-K síncronos por nivel

Estudiaremos ambos biestables simultáneamente toda vez que son muy parecidos y que el sincronismo sólo afecta al cuándo evolucionan los biestables, y no al qué hacen los biestables.

A continuación se muestran dos circuitos y tablas de verdad correspondientes a un R-S y a un J-K, ambos síncronos por nivel, figuras 7-17 y 7-18 y tablas 7-9 y 7-10, respectivamente.

Figura 7-17

Circuito de un biestable R-S síncrono por nivel.

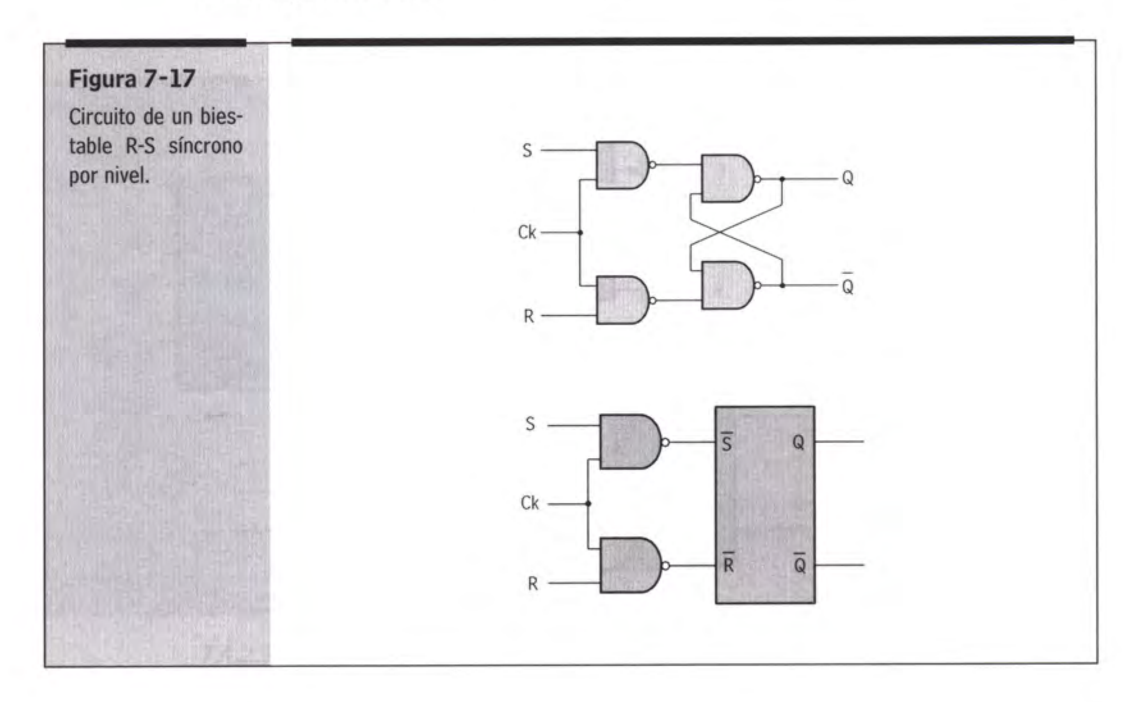


Tabla 7-9

Bloque y tabla de un R-S síncrono por nivel alto.

Ck	S	R	Q
1	1	0	1
1	0	1	0
1	0	0	Q_{t-1}
1	1	1	Prohibido
0	X	X	Q_{t-1}

S
Q
Ck
R - S
K
$\overline{Q}$

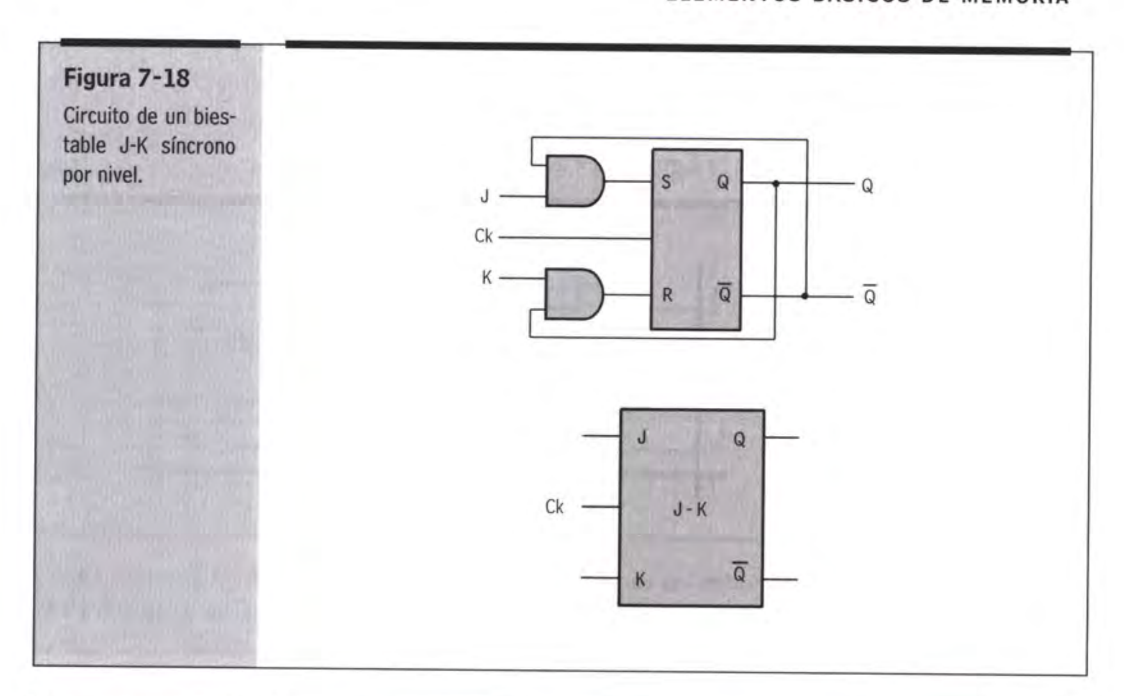


Figura 7-18
Circuito de un biestable J-K síncrono por nivel.

Tabla 7-10
Tabla de un J-K síncrono por nivel alto.

Ck	J	K	Q
1	1	0	1
1	0	1	0
1	0	0	Q_{t-1}
1	1	1	Prohibido
0	X	X	Q_{t-1}

Estudiemos el circuito del biestable R-S síncrono desde el álgebra de Boole:

1. Si Ck=0 entonces

$\overline{S} = \overline{Ck \cdot S} = 1$ y $\overline{R} = \overline{Ck \cdot R} = 1$;

Si $\overline{S} = 1$ y $\overline{R} = 1$ entonces $Q_t = Q_{t-1}$

2. Si Ck=1 entonces

$\overline{S} = \overline{Ck \cdot S} = \overline{S}$ y $\overline{R} = \overline{Ck \cdot R} = \overline{R}$

$\overline{S} = \overline{S}$ y $\overline{R} = \overline{R}$, y por tanto el biestable reacciona según la tabla del R-S asíncrono.

Resumiendo, el Ck se comporta como *una llave que abre o cierra* el paso de las entradas:

- Si el Ck=1 el biestable R-S se comporta como un asíncrono cuyas entradas son activas ahora por nivel alto.

- Si el Ck=0, este valor cierra el paso a los valores de S y R, y por tanto $\overline{S} = \overline{R} = 1$, manteniéndose con ello el valor del biestable. Es decir, aunque R y S varíen, el biestable *no escuchará* dichas variaciones.

El cronograma de la figura 7-19 aclara lo anterior.

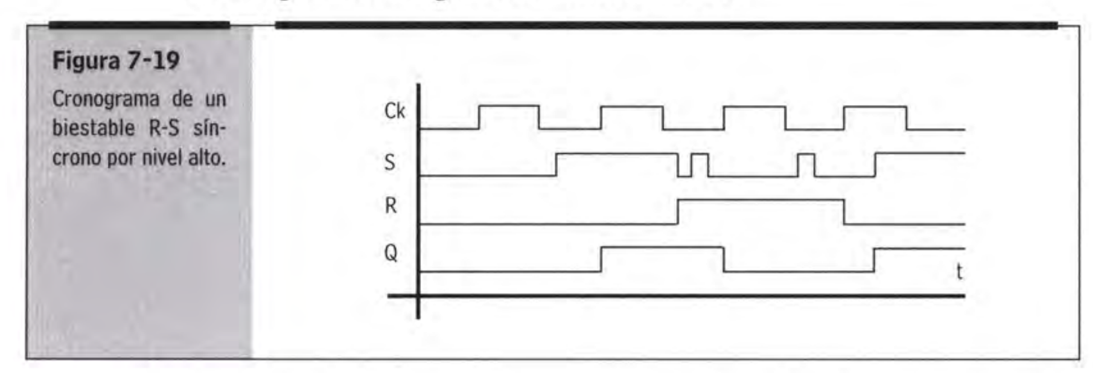


Figura 7-19
Cronograma de un biestable R-S síncrono por nivel alto.

Obsérvese que todos los cambios que se den en R y S cuando el Ck=0 no afectarán a la salida, y cómo cualquier cambio en R y S, estando Ck=1, afectará a la salida.

Respecto del J-K podríamos repetir el planteamiento anterior, ya que éste se basa en un R-S síncrono, pero lo dejamos como una actividad para el lector.

7.5.1.1. Biestables síncronos por nivel bajo

Si en vez de un biestable síncrono por nivel alto éste debiera serlo por nivel bajo, bastaría con invertir la señal de reloj. En este caso sólo afectarían a la salida aquellos cambios en la entrada que se produzcan cuando Ck=0.

En la figura 7-20 y la tabla 7-11 vemos los correspondientes circuitos y tablas de verdad para R-S y J-K con sincronismo por nivel bajo (el círculo del bloque significa nivel bajo).

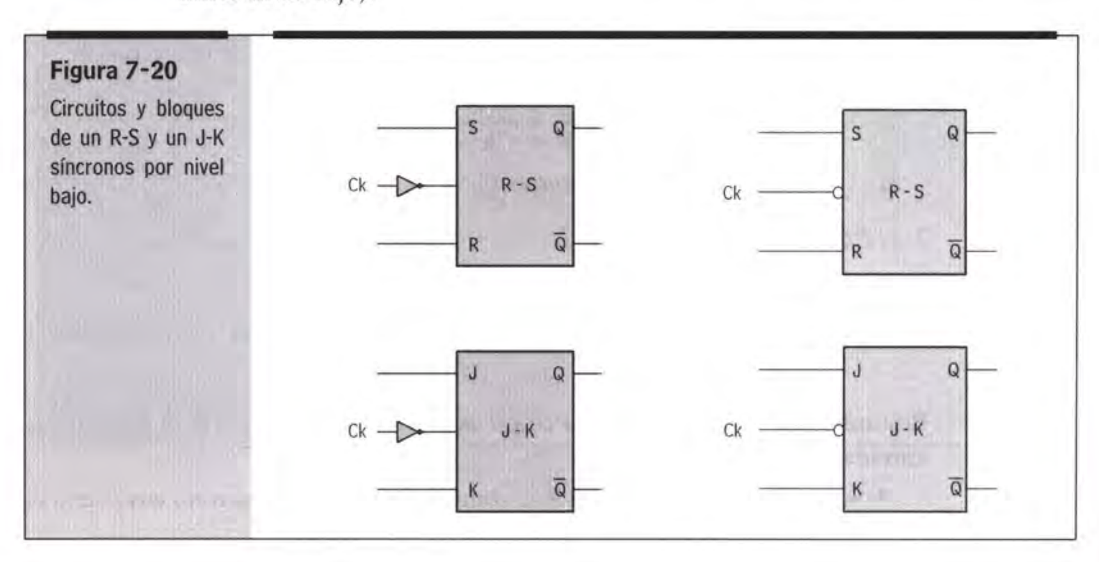


Figura 7-20
Circuitos y bloques de un R-S y un J-K síncronos por nivel bajo.

Tabla 7-11
Tabla de un R-S y de un J-K síncronos por nivel bajo.

Ck	S	R	Q_t
0	1	0	1
0	0	1	0
0	0	0	Q_{t-1}
0	1	1	Prohibido
1	X	X	Q_{t-1}

Ck	J	K	Q_t
0	1	0	1
0	0	1	0
0	0	0	Q_{t-1}
0	1	1	Prohibido
1	X	X	Q_{t-1}

El cronograma de la figura 7-21 corresponde a un biestable J-K síncrono por nivel bajo.

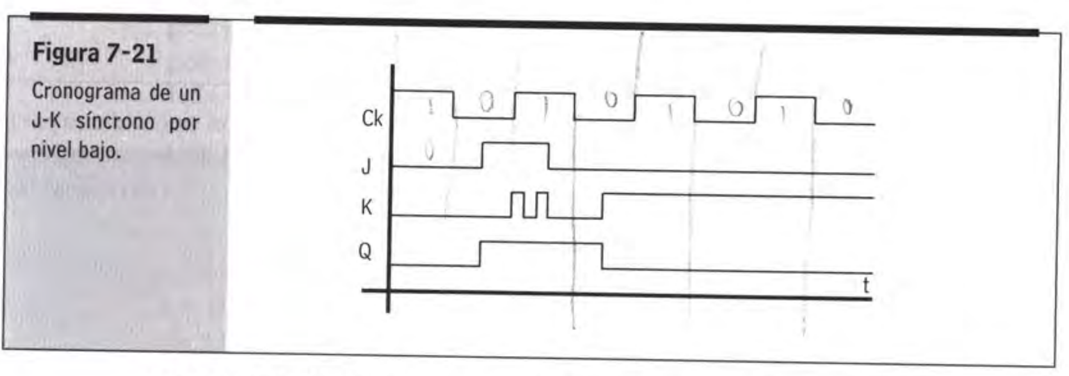


Figura 7-21
Cronograma de un J-K síncrono por nivel bajo.

7.5.1.2. Evolución de la indeterminación con el sincronismo

Anteriormente hemos visto cómo tanto el J-K como el R-S asíncronos podían tomar un valor desconocido, dejando al sistema indeterminado. La cuestión ahora es: ¿el sincronismo por nivel mejora o empeora la situación? La respuesta es que la empeora.

Veamos, en los asíncronos la indeterminación era posible a nivel teórico, pero imposible a nivel práctico, ya que nunca se iba a dar simultáneamente el paso de 11 a 00, o viceversa. Sin embargo, ahora la pérdida de sincronismo asegura esa simultaneidad.

La tabla 7-12 muestra el comportamiento de los biestables J-K y R-S cuando estando sus entradas a 1 el reloj pasa a 0. Vemos que en ambos casos la salida queda indeterminada, ya provenga del absurdo (R-S) o de las carreras (J-K).

El cronograma de la figura 7-22 aclara la tabla 7-12, y es válido para biestables J-K o R-S síncronos por nivel alto.

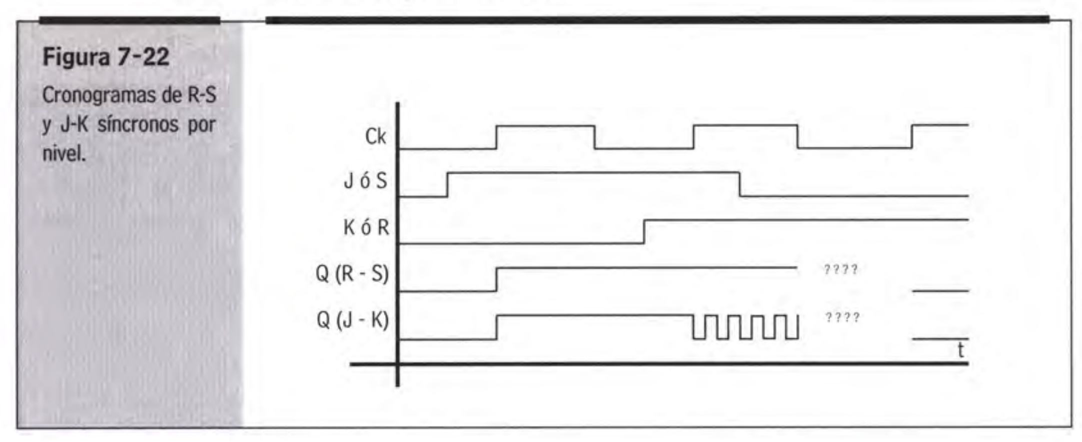


Figura 7-22

Cronogramas de R-S y J-K síncronos por nivel.

Tabla 7-12

Estudio de los estados de prohibición e indeterminación.

Reloj		Ent. Biestable	Ent. Asíncr.	Salida R-S	Salida J-K
1.	Ck=1	J=S=1	$\overline{S}=0$	Q= $\overline{Q}$ =1	carreras
		K=R=1	$\overline{R}=0$		
2.	Ck=0	J=S=1	$\overline{S}=1$	Q=??	Q =??
		K=R=1	$\overline{R}=1$	$\overline{Q}$=??	$\overline{Q}$ =??

Así pues, basta con que estando J=K=1 o R=S=1 se pierda el sincronismo para que la salida del biestable quede indeterminada. Y esta pérdida de sincronismo no es sólo posible, sino inherente al sistema. De todas formas, vemos que si las entradas vuelven a la normalidad, la salida también lo hace.

Los cronogramas de las figuras 7-23 y 7-24 muestran distintas situaciones prohibidas e indeterminadas.

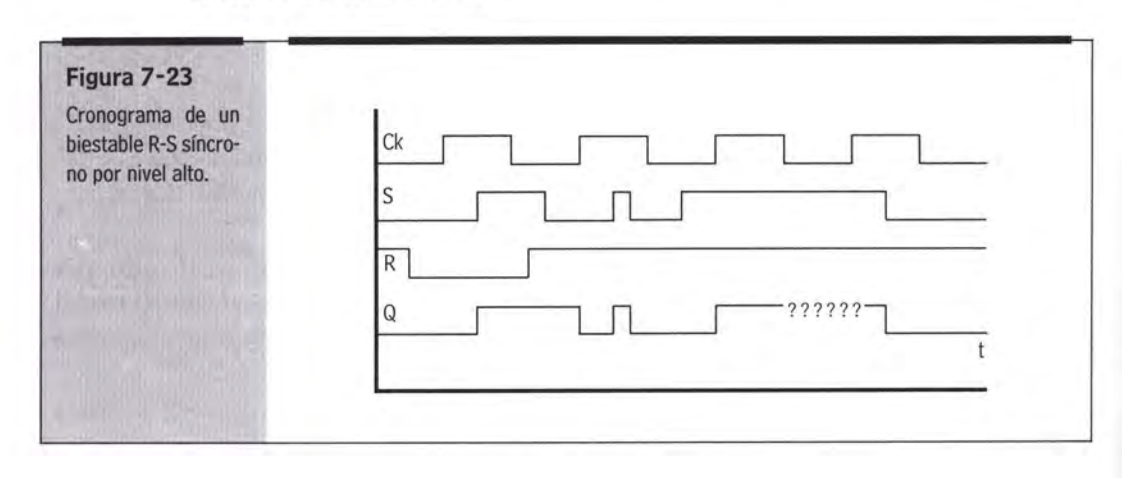
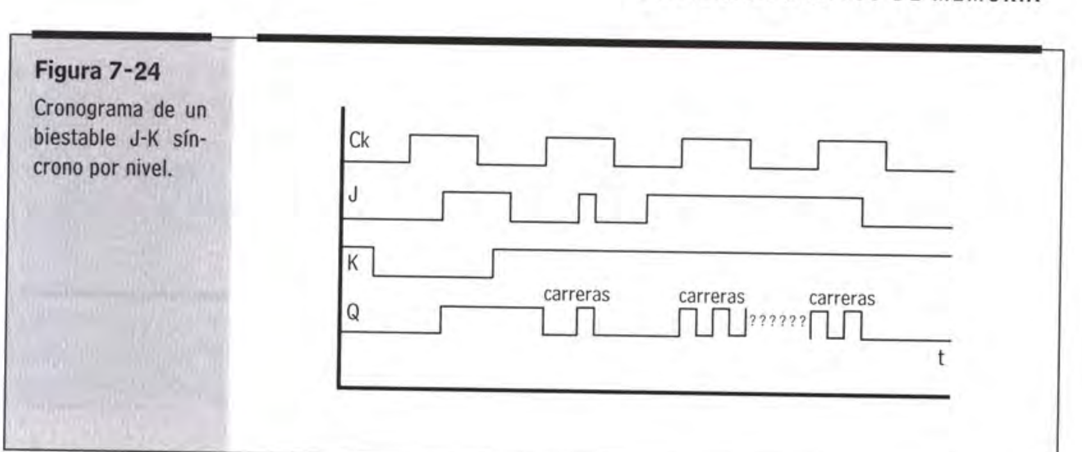


Figura 7-23

Cronograma de un biestable R-S síncrono por nivel alto.

Figura 7-24
Cronograma de un biestable J-K síncrono por nivel.



Podemos afirmar que el sincronismo por nivel degrada claramente y por igual el comportamiento de los biestables R-S y J-K. Si vamos a usar biestables síncronos por nivel debemos asegurarnos de que no se producirán situaciones prohibidas, o que éstas serán despreciables. El biestable tipo D, que vemos en 7.5.2, elimina la indeterminación y permite su uso con sincronismo por nivel.

7.5.1.3. Ecuaciones de los biestables R-S y J-K síncronos por nivel

Las tablas de verdad de 7-11 para ambos biestables son idénticas si no contemplamos las situaciones prohibidas. Para el caso de sincronismo por nivel alto podemos rediseñar y reimplementar los biestables siguiendo las ecuaciones obtenidas del V-K de la tabla de verdad 7-13.

Tabla 7-13
Tabla de los biestables R-S y J-K síncronos por nivel alto.

Ck	J o S	K o R	Q_{t-1}	Q_t
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	X
1	1	1	1	X
0	X	X	0	0
0	X	X	1	1

Las ecuaciones resultantes a implementar son:

$$Q_t = Q_{t-1} \cdot \overline{R} + S \cdot Ck + \overline{Ck} \cdot Q_{t-1}$$
$$Q_t = Q_{t-1} \cdot \overline{K} + J \cdot Ck + \overline{Ck} \cdot Q_{t-1}$$

Siempre que $R \cdot S = 0$ y $J \cdot K = 0$, es decir, que nunca $J = K = 1$ o $R = S = 1$ (ver condiciones libres en la tabla 7-13).

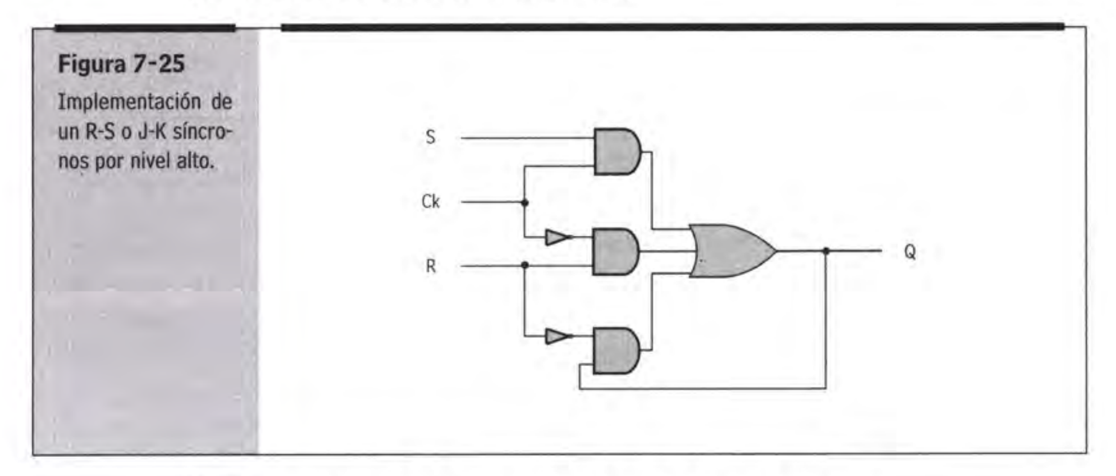


Figura 7-25
Implementación de un R-S o J-K síncronos por nivel alto.

7.5.1.4. Resumen de los biestables R-S y J-K síncronos por nivel

En un biestable síncrono por nivel, si el Ck no está activo la salida no varía, y si el reloj está activo la salida varía con la entrada en el modo ya estudiado para los biestables R-S y J-K asíncronos. El sincronismo no afecta ni al qué ni al cómo, afecta al cuándo evolucionan las salidas frente a las entradas.

Por otra parte, la sincronía conlleva la clara posibilidad de que el biestable alcance estados indeterminados, situación ésta que era casi imposible en los asíncronos. Esta razón hace que los J-K y R-S síncronos por nivel no sean muy utilizados.

7.5.2. Biestable D síncrono por nivel

El biestable tipo D es muy utilizado, su función es copiar la salida en la entrada, si el reloj lo permite. Es decir, si el biestable está sincronzado la salida *sigue* a la entrada, pero si el biestable no está sincronizado la salida mantiene (memoriza) su valor aunque cambie la entrada. Su esquema y tabla de verdad se muestran en la figura 7-26 y la tabla 7-14, respectivamente.

Tabla 7-14
Tabla de un D síncrono por nivel alto.

Ck	D	Qt
1	1	1
1	0	0
0	X	Q_{t-1}

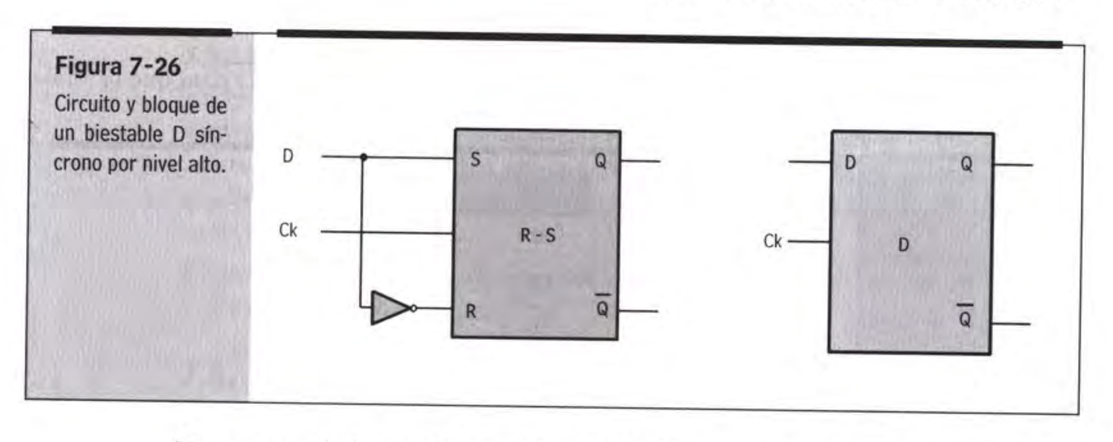


Figura 7-26
Circuito y bloque de un biestable D síncrono por nivel alto.

Vemos que el circuito lógico del biestable D es muy sencillo. Tiene una única entrada, y en él no se puede dar la situación prohibida, ni por tanto la indeterminada. Por otro lado, el biestable D no tiene función de memoria como operación de las entradas, sólo memoriza cuando pierde el sincronismo.

En muchos casos, en un biestable D síncrono por nivel el reloj no es tal, sino que es una simple línea de control, llamada ENABLE (como si fuera una *llave*). En este caso el biestable recibe preferentemente el nombre de *latch*.

En los cronogramas de las figuras 7-27 y 7-28 vemos la evolución de un biestable D síncrono por nivel. El primero con señal de reloj Ck y el segundo con línea de control ENABLE (E).

Figura 7-27
Cronograma de un biestable D síncrono por nivel alto.

Ck
D
Q
t

Figura 7-28
Cronograma de un D síncrono por nivel como latch.

E
D
Q
t

7.5.2.1. Uso y ecuaciones de un biestable D síncrono por nivel

La tabla 7-15 determina qué valores debe tomar la entrada D para que la salida actual cambie al nuevo valor deseado, siendo Ck=1.

Tabla 7-15
Tabla de un D síncrono por nivel alto.

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Si planteamos la tabla de verdad del biestable D para rediseñarlo obtendremos la tabla 7-16.

Tabla 7-16
Tabla de verdad de un D síncrono por nivel alto.

Ck	D	Q_t	Q_{t+1}
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1

Del correspondiente V-K, la ecuación resultante a implementar es:

$$Q_{t+1} = D \cdot Ck + \overline{Ck} \cdot Q_t$$

Esta ecuación tiene un riesgo estático que es eliminado si añadimos $Q_t \cdot D$

$$Q_{t+1} = D \cdot Ck + \overline{Ck} \cdot Qt + D \cdot Q_t$$

El esquema resultante para la primera ecuación se muestra en la figura 7-29.

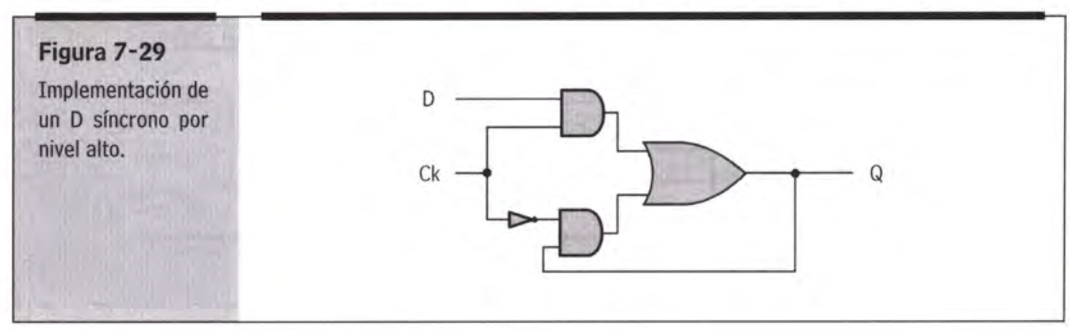


Figura 7-29
Implementación de un D síncrono por nivel alto.

7.5.2.2. Resumen del biestable D síncrono por nivel

En el biestable D síncrono por nivel la salida toma el valor de D si el reloj está activo, en caso contrario la salida no varía. El biestable D *copia* la entrada en la salida mientras el biestable esté sincronizado. El biestable D muchas veces es usado como *cerrojo*, en cuyo caso la línea de Ck pasa a ser una línea no periódica de control, ENABLE.

El biestable D, por su implementación, no tiene problemas de indeterminación y es muy utilizado.

7.6. Biestables Maestro/Esclavo

Los biestables tipo M/S (Master/Slave) están a medio camino entre el sincronismo por nivel y el sincronismo por flanco. La filosofía del biestable M/S es dividir su modo de operación en dos fases aisladas entre sí en el tiempo. El biestable maestro recoge las entradas y obtiene una primera salida. Esta primera salida es recibida como entrada por el esclavo, que simplemente la copia en la salida. Puede parecer que el esclavo no hace nada -y así es a nivel lógico- pero su misión es separar en el tiempo la lectura de las entradas de la escritura de las salidas, y así eliminar las realimentaciones originadoras de las carreras. Resumiendo, el maestro *escucha y piensa*, mientras que el esclavo *sólo copia*.

Este aislamiento entre la captura de la entrada y la escritura de la salida evita las realimentaciones sin control, situación que beneficia claramente al J-K, y no tanto al R-S.

7.6.1. Biestable J-K tipo M/S

Siguiendo la filosofía anterior, podemos ver en la figura 7-30 que el maestro capta las entradas en el nivel alto de reloj, y que el esclavo escribe en la salida durante el nivel bajo del reloj lo que le transmite el maestro.

El biestable maestro es un J-K y el esclavo es un R-S en modo de copia (tipo D).

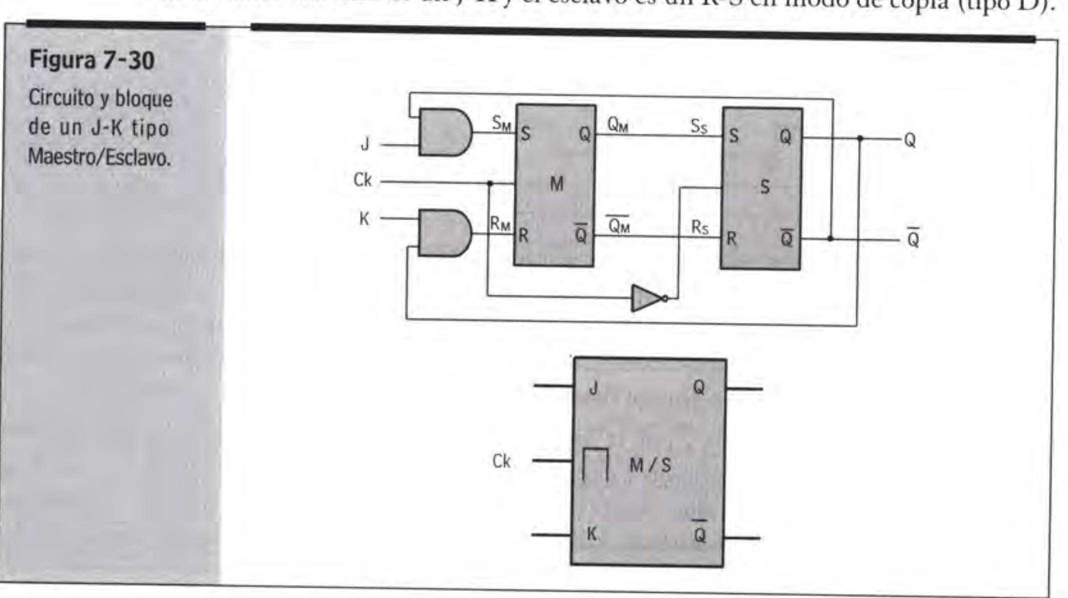


Figura 7-30
Circuito y bloque de un J-K tipo Maestro/Esclavo.

En la tabla de verdad 7-17 es complicado representar el Ck, puesto que para Ck =1 es activo el maestro y para Ck=0 lo es el esclavo. Para entender su comportamiento, que describiremos a continuación, basta entender que para Ck=1 tenemos un J-K y para Ck=0 tenemos un R-S, ambos activos por nivel. En los diferentes casos que se muestran a continuación el lector debe tener presente la tabla del R-S síncrono por nivel.

1. Si J=0 y K=0. MANTENIMIENTO

Cuando Ck=1 entonces SM=RM=0 luego no varían ni QM, ni SS, ni RS.

Cuando Ck=0 como no han cambiado ni SS, ni RS entonces $Q_t=Q_{t-1}$.

2. Si J=1 y K=0. PUESTA A 1

Cuando Ck=1: SM=$\overline{Q}$ = X y RM=0, luego QM=1 y por tanto SS=1 y RS=0.

Cuando Ck=0 como SS=1 y RS=0 la salida se pone a 1, Q=1.

3. Si J=0 y K=1. PUESTA A 0

Cuando Ck=1: SM=0 y RM=Q = X, luego QM=0 y por tanto SS=0 y RS=1.

Cuando Ck=0 como SS=0 y RS=1 la salida se pone a 0, Q=0.

4. Si J=1 y K=1. BASCULAMIENTO

Cuando Ck=1: SM=$\overline{Q}$ y RM=Q, luego $QM_t=SS=\overline{Q_{t-1}}$ y $\overline{QM_t}=RS=Q_{t-1}$ (el basculamiento sólo se da una vez).

Cuando Ck=0 como $SS=\overline{Q_{t-1}}$ y $RS=Q_{t-1}$ entonces Q bascula, $Q_t=\overline{Q_{t-1}}$ y $\overline{Qt}=Q_{t-1}$.

De lo anterior, y sin representar el reloj, obtenemos la tabla de verdad 7-17.

Tabla 7-17

Tabla de un J-K M/S.

J	K	Q_t	Operación
0	0	Q_{t-1}	Memoria
1	0	1	Puesta a 1
0	1	0	Puesta a 0
1	1	$\overline{Q_{t-1}}$	Basculamiento

Recordemos que si J=K=1, aunque QM invierte su valor, sólo lo hace una vez, ya que el efecto de QM no pasa a Q hasta que Ck=0, pero en dicho momento el cambio de Q ya no afecta a QM (ya no está sincronizado). Así, este aislamiento entre etapas hace que el fenómeno de carreras se convierta en una única carrera, o sea, un simple basculamiento.

Los símbolos ⊓ y ⊔ de la figura 7-31 indican, respectivamente, que el maestro es activo por nivel alto y el esclavo por bajo, y que el maestro es activo por bajo y el esclavo por alto. También hay que reseñar que para un J-K tipo ⊓ aunque el efecto de la entrada es real cuando Ck=1, este efecto no estará presente en la

salida hasta que el Ck pase a 0, o mejor aún, cuando se haya producido el flanco descendente Ck=$\downarrow$. Resumiendo, no debemos acceder a la salida hasta que se produzca el flanco adecuado.

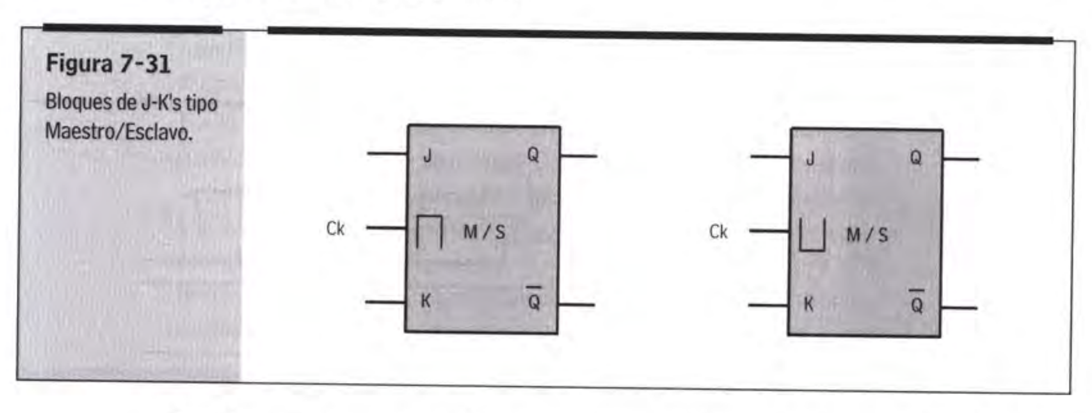


Figura 7-31
Bloques de J-K's tipo Maestro/Esclavo.

Por último, destaquemos algunos aspectos particulares del biestable J-K M/S:

- Su circuito es más complicado, ya que utiliza dos biestables R-S.
- Necesita todo un nivel y un flanco para obtener la salida.
- Las entradas no deben variar durante el nivel activo del maestro, y si lo hacen la salida sólo seguirá a la última variación.
- Si se producen *glitches* durante la actividad del maestro nos encontraremos ante un problema conocido como de captación de unos, produciendo un mal funcionamiento del biestable.

En el cronograma de la figura 7-32 observamos el comportamiento de dos biestables J-K M/S. El primero es tipo ⊓ (maestro por nivel alto y esclavo por flanco descendente $\downarrow$) y el segundo es tipo ⊔ (maestro por nivel bajo y esclavo por flanco ascendente $\uparrow$).

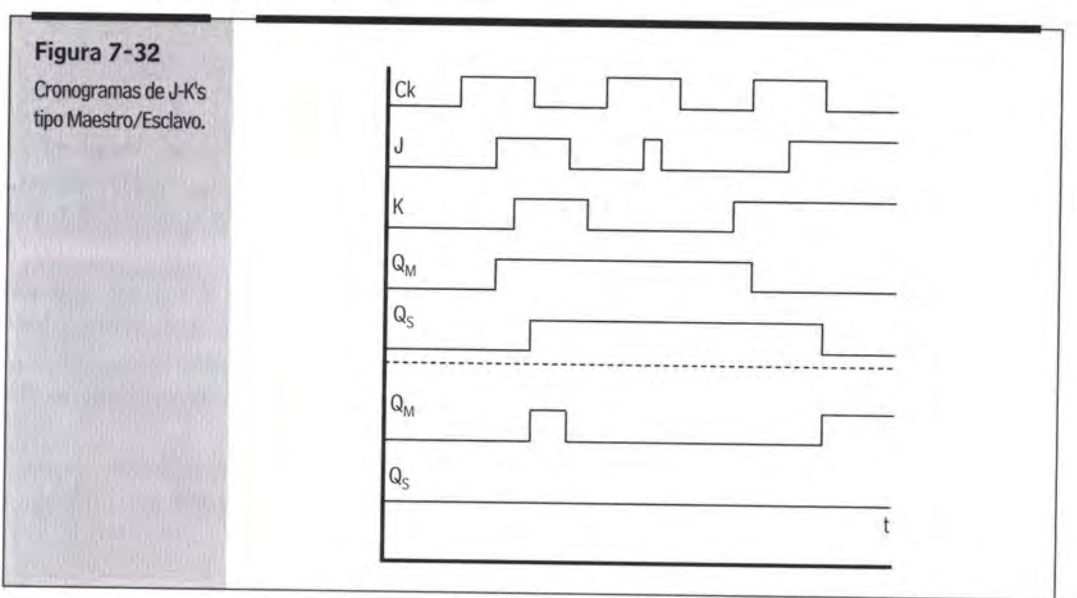


Figura 7-32
Cronogramas de J-K's tipo Maestro/Esclavo.

Observando el cronograma anterior, vemos cómo el esclavo sigue al maestro con un retardo de medio ciclo de reloj, excepto si las señales J y/o K cambian durante el nivel activo del maestro. En el cronograma de la figura 7-33 se respeta esta restricción (J y K no varían), y observamos el comportamiento regular del biestable JK M/S ⊓.

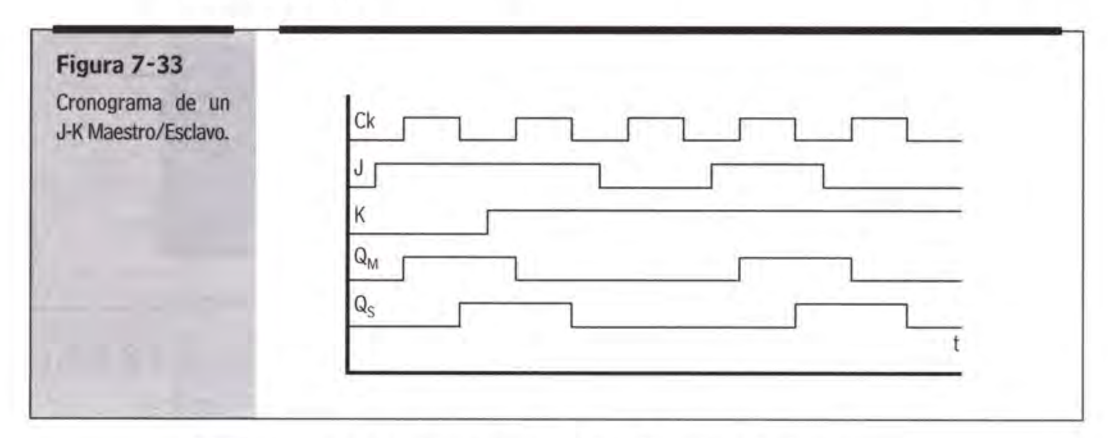


Figura 7-33
Cronograma de un J-K Maestro/Esclavo.

7.6.2. Resumen del biestable J-K tipo Maestro/Esclavo

La filosofía Maestro/Esclavo, con ambas etapas aisladas, elimina el problema de carreras del biestable J-K asíncrono, pues ahora, cuando J=K=1 se produce una única carrera, llamada basculamiento. El resto de combinaciones de la entrada hace evolucionar al biestable J-K como siempre.

En cuanto a la evolución en el tiempo del J-K M/S, hay que resaltar que la salida correspondiente a la entrada no estará disponible hasta que en el reloj se produzca el flanco adecuado (descendente para ⊓ y ascendente para ⊔).

Como única restricción cabe destacar que las entradas no deben variar durante el nivel activo del maestro, para así evitar la captación de unos.

7.7. Biestables síncronos por flanco

En los biestables síncronos por nivel existía el problema de los estados prohibidos e indeterminados. Para los J-K M/S desaparecía el problema de las carreras, pero aparecía el de captación de unos. Este último problema quedará resuelto con los biestables síncronos por flanco.

La filosofía se basa en que el biestable esté sincronizado *-a la escucha-* sólo en el instante en que se produzca un flanco o transición, ya sea ascendente o descendente. De esta forma, como el flanco dura un infinitésimo (exagerando), sólo es capturado un único valor de las entradas, evitando así el problema de captación de unos.

Los biestables síncronos por flanco son los más utilizados en sistemas secuenciales síncronos, y en muchos casos son los únicos que pueden ser utilizados. Son los llamados flip-flop.

7.7.1. Detección de flancos

Inicialmente, y de forma válida para todos los biestables, veremos cómo obtener y manejar los flancos de un reloj. La idea es no cambiar el biestable original, sino modificar el reloj. Así, si el reloj original era simétrico, en el reloj modificado sólo quedarán a nivel activo los flancos elegidos como activos, es decir, el nuevo reloj estará siempre inactivo excepto para los flancos que sean activos en el reloj original.

En la figura 7-34 vemos cómo modificar el reloj original Ck para obtener uno nuevo, Ck', que sólo contenga flancos ascendentes o descendentes. El circuito se basa en el pequeño retardo (tr) que el inversor introduce, así el producto de Ck · $\overline{Ck}$ no dará siempre 0, sino que durante un breve espacio de tiempo (el retardo tr del inversor) será 1. Este 1 de corta duración se asemeja mucho a un flanco ascendente, aunque realmente no lo sea. El comportamiento del circuito es descrito esquemáticamente en los cronogramas de las figuras 7-35 y 7-36.

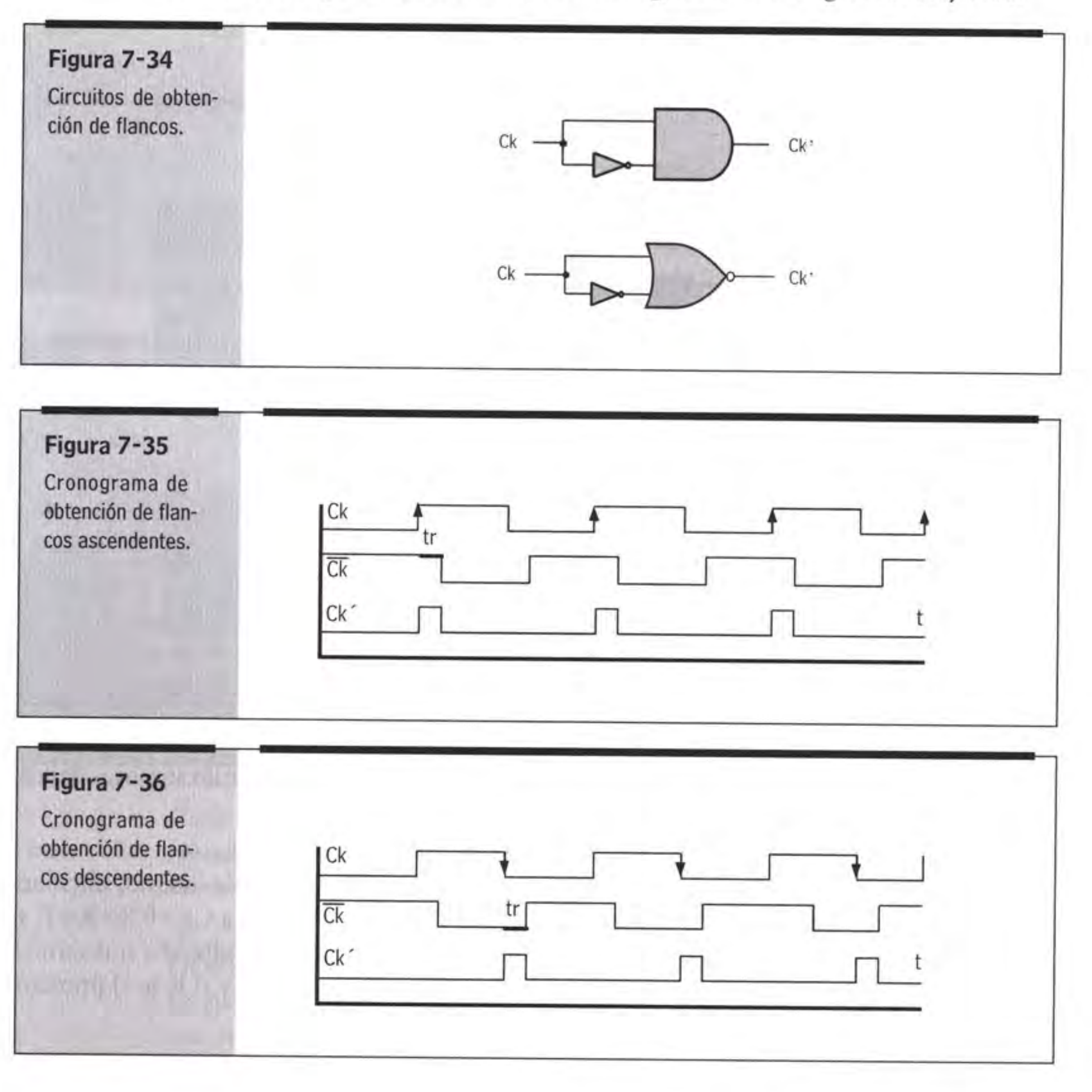


Figura 7-34
Circuitos de obtención de flancos.

Figura 7-35
Cronograma de obtención de flancos ascendentes.

Figura 7-36
Cronograma de obtención de flancos descendentes.

Obsérvese que aunque en el segundo caso se trate de flancos descendentes, éstos se convierten en niveles altos. Queda como ejercicio para el lector obtener el reloj modificado para biestables síncronos por nivel bajo, es decir, que el flanco elegido como activo se presente como un nivel bajo.

7.7.2. Biestable R-S síncrono por flanco

El análisis del R-S síncrono por flanco sólo tiene interés teórico, ya que en la práctica no puede ser utilizado por el mal comportamiento frente a la situación prohibida.

Toda vez que tenemos el nuevo reloj modificado, Ck', y un R-S síncrono por nivel alto, el diseño se basa en utilizar Ck' y no Ck como reloj del R-S (figura 7-37). De esta manera, el R-S estará activo en unos niveles de duración mínima, que son coincidentes con los flancos ascendentes o descendentes del reloj Ck original. Resumiendo, a nivel teórico el biestable R-S sigue siendo síncrono por nivel, pero a nivel práctico lo es por flanco.

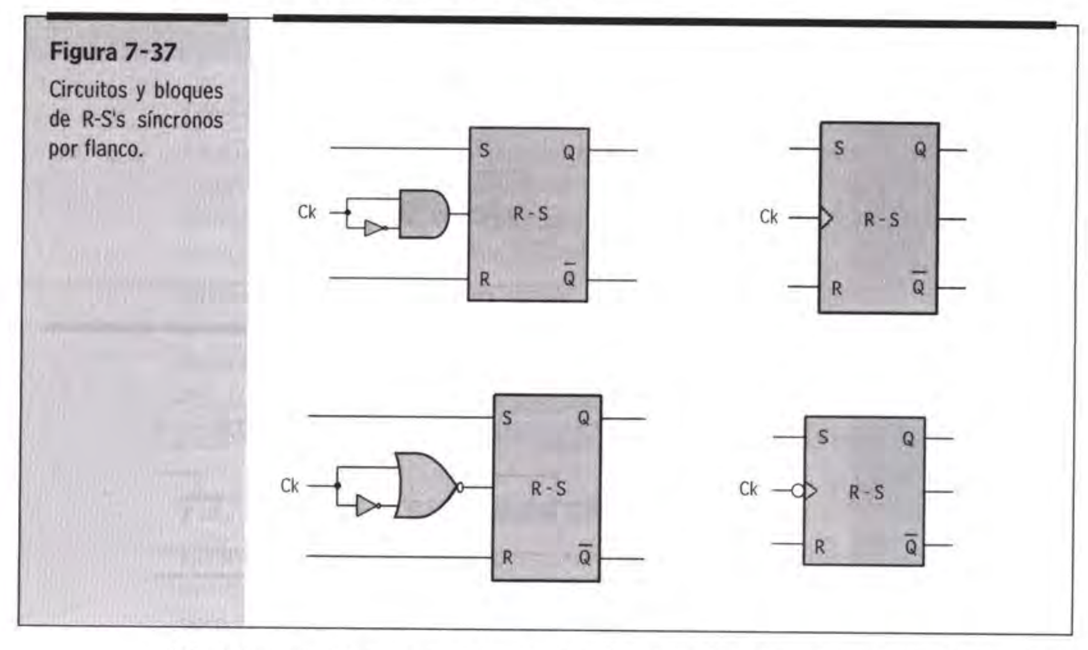


Figura 7-37
Circuitos y bloques de R-S's síncronos por flanco.

El triángulo que se encuentra dentro del símbolo R-S indica sincronía por flanco; un círculo delante significa flanco descendente, y sin círculo significa flanco ascendente.

Destacar en la tabla de verdad 7-18 el caso S=R=1. En este caso, la salida siempre va a pasar a indeterminada, porque visto desde el biestable interno, síncrono por nivel, resulta que las entradas pasan de Ck=1 S=R=1 a Ck=0 S=R=1, y ya vimos que esta pérdida de sincronía estando S=R=1 conllevaba indeterminación. Los cronogramas de las figuras 7-38 y 7-39 describen al R-S; el primero no contempla situaciones prohibidas, el segundo sí.

Tabla 7-18

Tabla de R-S síncrono por flanco descendente.

Ck	S	R	Q_t	Operación
↓	0	0	Q_{t-1}	Mantenimiento
↓	1	0	1	Puesta a 1
↓	0	1	0	Puesta a 0
↓	1	1	??	Prohibido e Indeter.

Figura 7-38

R-S's síncronos por flanco ascendente y descendente.

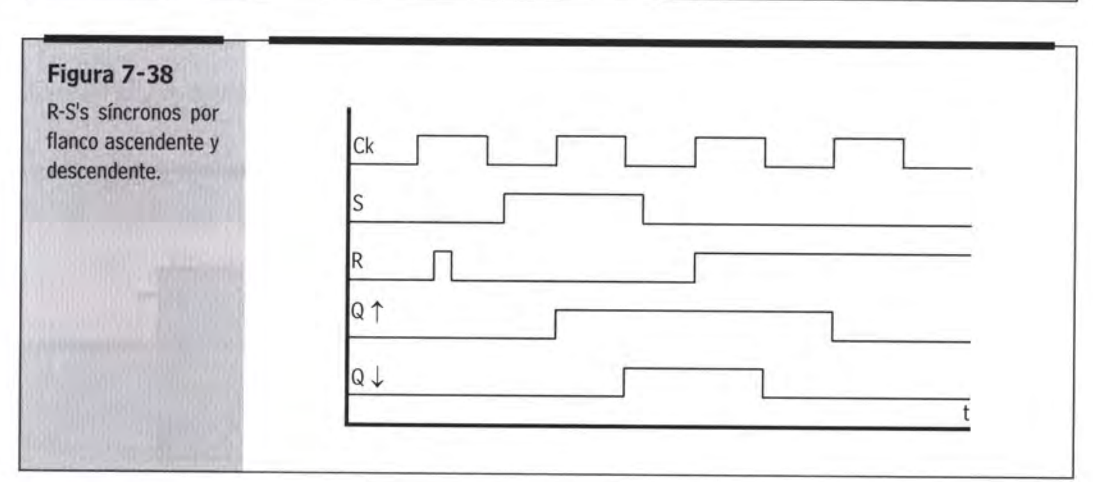
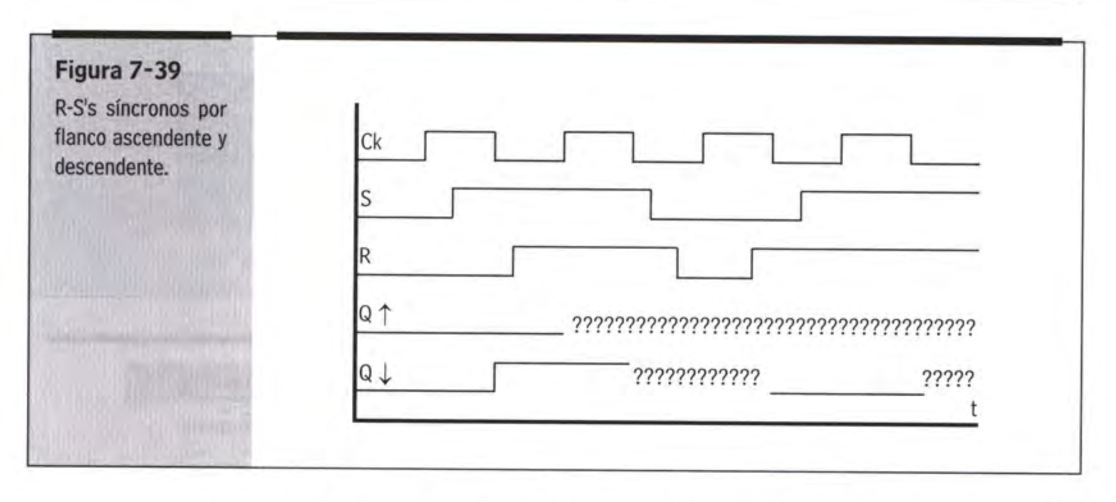


Figura 7-39

R-S's síncronos por flanco ascendente y descendente.



Cabe concluir que la sincronía ha degradado el modo de funcionamiento del biestable R-S, ya que la indeterminación ha pasado de posible pero improbable en el asíncrono, a casi inevitable en el síncrono por flanco.

7.7.3. Biestable J-K síncrono por flanco

La filosofía seguida es idéntica a la planteada para el R-S, con la diferencia de que en el J-K la sincronía por flanco mejora su funcionamiento (figura 7-40).

Como ya vimos, el modo Maestro/Esclavo eliminaba las carreras, pero presentaba el problema de captación de unos. Con la sincronía por flanco también desaparecen las carreras, ya que aunque éstas se puedan dar -pues el J-K interno es síncrono por nivel- la duración del nivel activo del Ck' es tan corto que para cuando se realimente la salida con la intención de dar lugar a carreras, el biestable J-K ya no estará sincronizado, y por tanto sólo se produce una inversión: la primera, como en el J-K M/S. Además, el problema de captación de unos queda solventado, puesto que la duración del nivel activo es tan corta que el biestable sólo tiene tiempo de captar un único valor de las entradas, ignorando posibles cambios y *glitches*.

La figura 7-40 y la tabla 7-19 son, respectivamente, el circuito lógico y la tabla de verdad de un J-K síncrono por flanco.

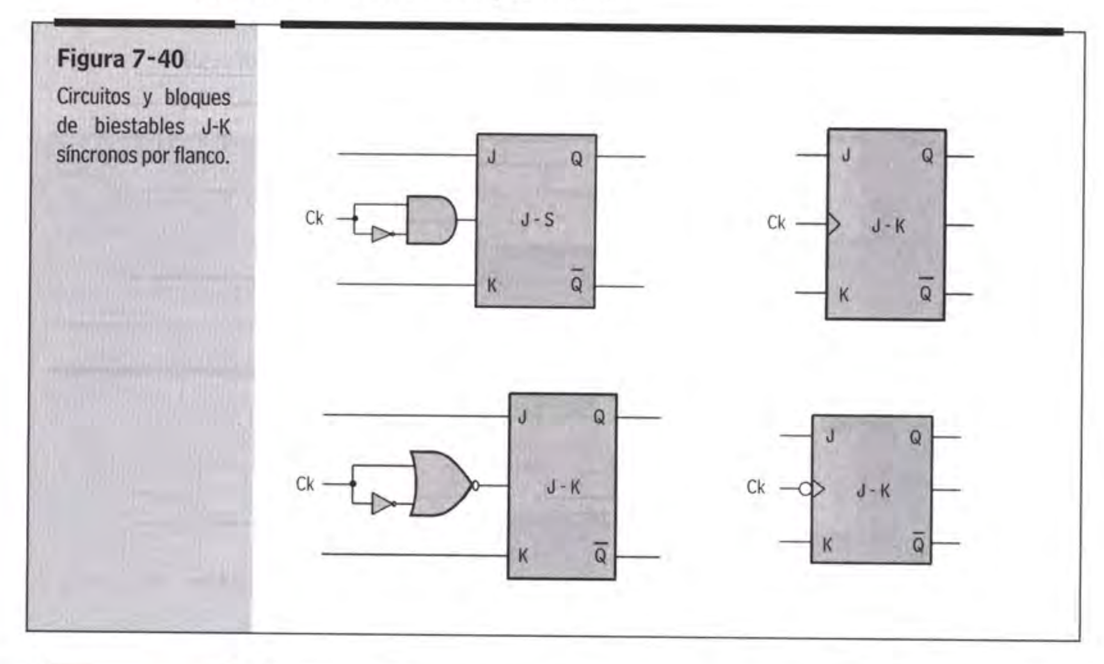


Figura 7-40
Circuitos y bloques de biestables J-K síncronos por flanco.

Tabla 7-19
Tabla de J-K síncrono por flanco descendente.

Ck	J	K	Qt	Operación
↓	0	0	Q_{t-1}	Memoria
↓	1	0	1	Puesta a 1
↓	0	1	0	Puesta a 0
↓	1	1	$\overline{Q_{t-1}}$	Basculamiento

En la figura 7-41 se muestran los diagramas de transición de estado de sendos biestables J-K y R-S, ambos activos por flanco ascendente (arbitrariamente, el símbolo $\overline{\uparrow}$ indica que el reloj no presenta un flanco activo).

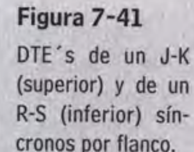
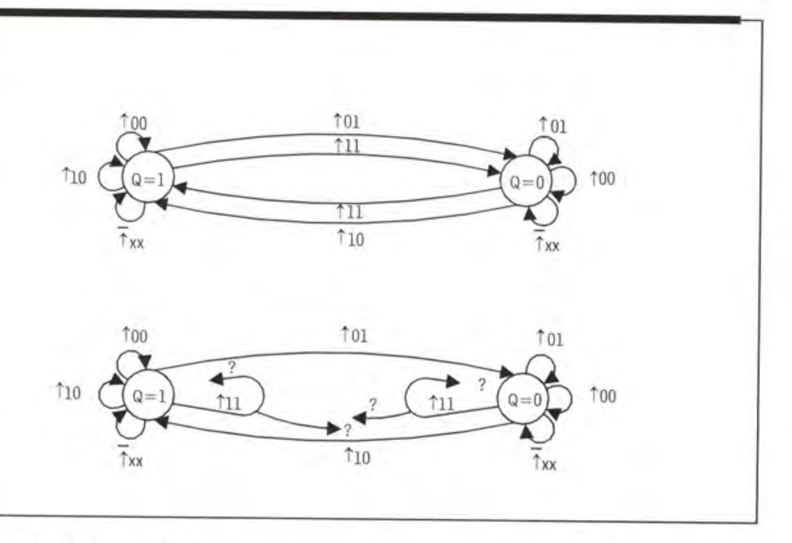


Figura 7-41

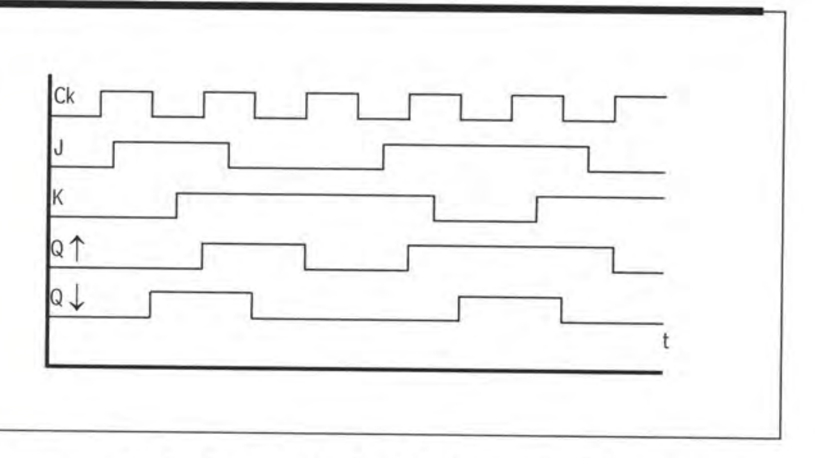
DTE´s de un J-K (superior) y de un R-S (inferior) síncronos por flanco.



El cronograma de la figura 7-42 muestra el funcionamiento de un J-K síncrono, tanto por flanco ascendente, como descendente.

Figura 7-42

Cronograma de J-K's síncronos por flanco.



Antes de dar por concluido el estudio de los J-K síncronos por flanco es interesante comparar éstos con el J-K M/S:

- Si ni J ni K variaran durante el nivel activo del maestro, entonces un J-K M/S ⊓ presentaría la misma salida que un J-K por flanco descendente. Lo mismo se puede decir entre el J-K M/S ⊔ y el J-K por flanco ascendente. Es decir, el flanco del esclavo establece la semejanza.
- Si J y/o K variaran durante el nivel activo del maestro, no tendría por qué darse la igualdad entre la salida del tipo M/S y la del síncrono por flanco.

Los cronogramas de las figuras 7-43 y 7-44 muestran la semejanza entre un J-K por flanco ascendente y un J-K M/S ⊔, y entre un biestable J-K activo por flanco descendente y un J-K M/S ⊓.

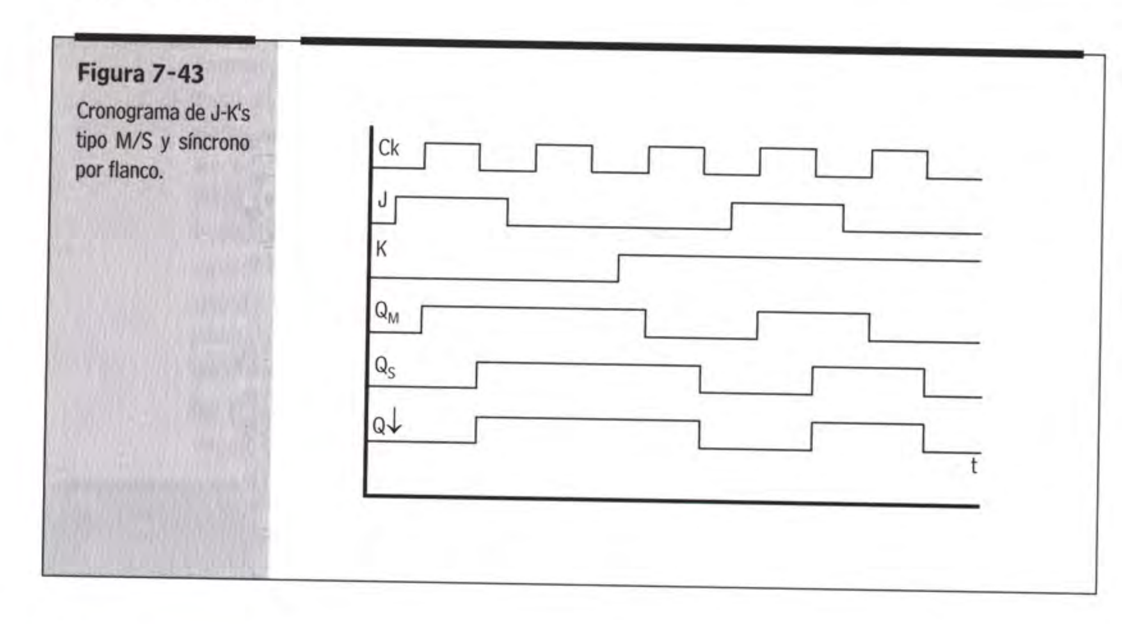


Figura 7-43
Cronograma de J-K's tipo M/S y síncrono por flanco.

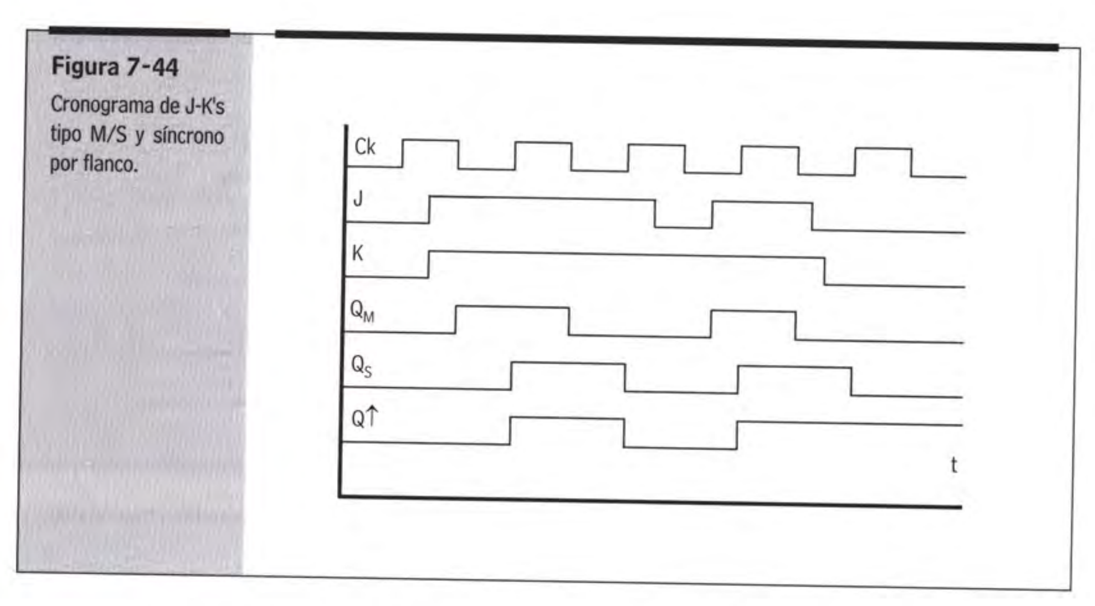


Figura 7-44
Cronograma de J-K's tipo M/S y síncrono por flanco.

Otro tipo de biestable J-K, híbrido entre el el tipo M/S y el activo por flanco, se denomina J-K M/S con cierre de datos. En este biestable el maestro es activo por flanco y el esclavo sigue siéndolo por nivel. Su uso está restringido a sistemas secuenciales complejos con problemas de retardo en la línea de reloj.

Volviendo al J-K, podemos obtener la ecuación booleana de un biestable J-K síncrono por flanco a partir de su tabla de verdad 7-20.

Tabla 7-20
Tabla de verdad de un J-K síncrono por flanco ascendente.

Ck	J	K	Q_{t-1}	Q_t
↑	0	0	0	0
↑	0	0	1	1
↑	0	1	0	0
↑	0	1	1	0
↑	1	0	0	1
↑	1	0	1	1
↑	1	1	0	1
↑	1	1	1	0

Del V-K, la ecuación resultante a implementar es:

$$Q_t = J \cdot \overline{Q_{t-1}} + \overline{K} \cdot Q_{t-1}, \text{ si } Ck=\uparrow$$

La tabla 7-21 muestra el modo en que hay que excitar a las entradas J y K para que las salidas actuales cambien en el modo deseado. Por ejemplo, si $Q_t=0$ y deseamos que cambie a 1 en el próximo flanco, $Q_{t+1}=1$, entonces J debe ser 1 y K cualquier valor (J=1 y K=X), etc. Esta tabla es muy útil en el capítulo 10 a la hora de diseñar con biestables J-K.

Tabla 7-21
Tabla de excitación de un J-K síncrono por flanco.

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

7.7.4. Biestable D síncrono por flanco

El biestable D síncrono por flanco se carga con el valor presente en la entrada en el momento del flanco activo. Veamos su implementación (figura 7-45), tabla de verdad (tabla 7-22), diagrama de transición de estados (figura 7-45) y cronograma (figura 7-46).

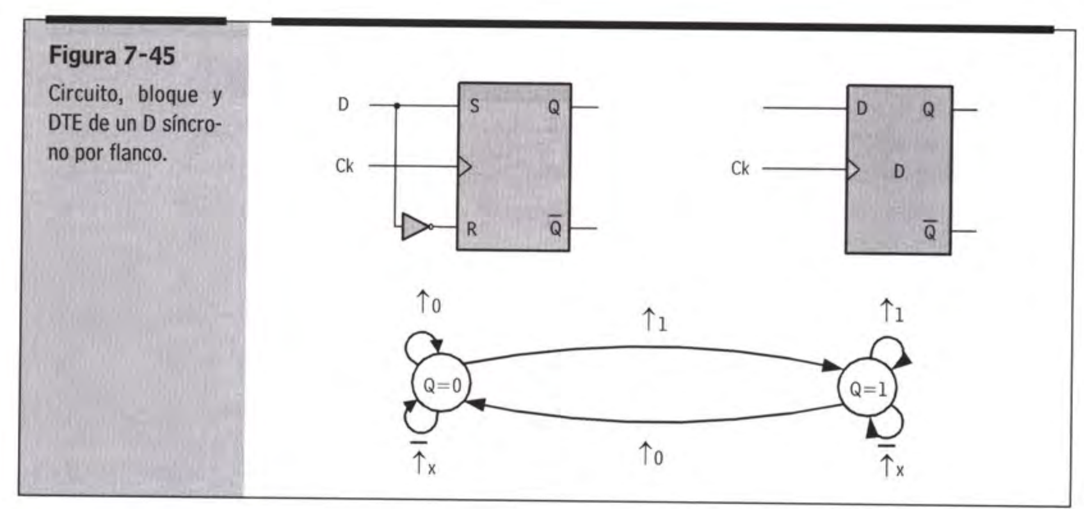


Figura 7-45
Circuito, bloque y DTE de un D síncrono por flanco.

Tabla 7-22
Tabla de funcionamiento de un D síncrono por flanco.

Ck	D	Qt	Operación
↓	1	1	Puesta a 1
↓	0	0	Puesta a 0

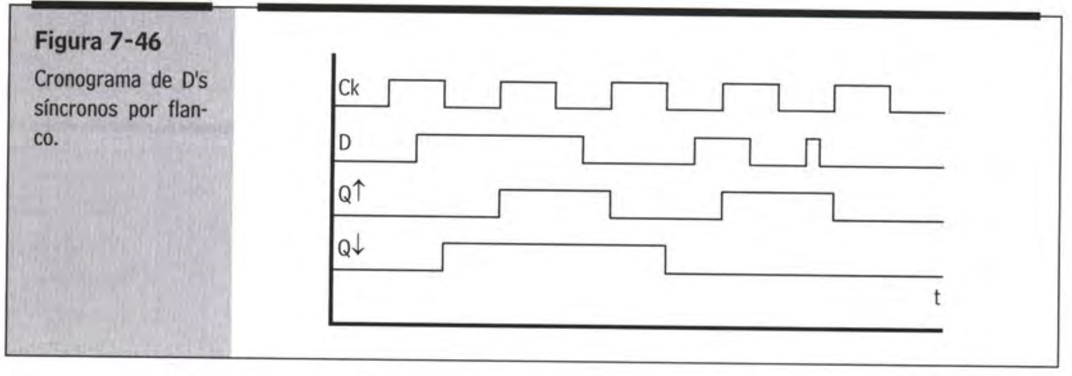


Figura 7-46
Cronograma de D's síncronos por flanco.

Podemos obtener la ecuación booleana de un biestable D síncrono por flanco a partir de la tabla 7-23.

Tabla 7-23
Tabla de un biestable D síncrono por flanco.

Ck	D	Q_{t-1}	Q_t
↑	0	0	0
↑	0	1	1
↑	1	0	0
↑	1	1	1

Del V-K, la ecuación resultante a implementar es:

$$Q_t = D \text{ si } Ck=\uparrow$$

Si lo que queremos es saber cómo excitar la entrada para conseguir la salida deseada, debemos contemplar la tabla 7-24, que en resumen dice que lo mismo que se desee en la salida es lo que se debe colocar en la entrada.

Tabla 7-24
Tabla de excitación de un D síncrono por flanco.

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Antes de abandonar el sincronismo por flanco cabe decir que el circuito de la figura 7-34 de captación de flancos sólo tiene validez teórica. Es decir, los biestables comerciales no funcionan así. La figura 7-47 muestra un biestable D síncrono por flanco; podemos decir que éste sí que es un *auténtico* flip-flop.

A continuación estudiaremos el biestable D síncrono por flanco ascendente de la figura 7-47 basado en la captación de flancos.

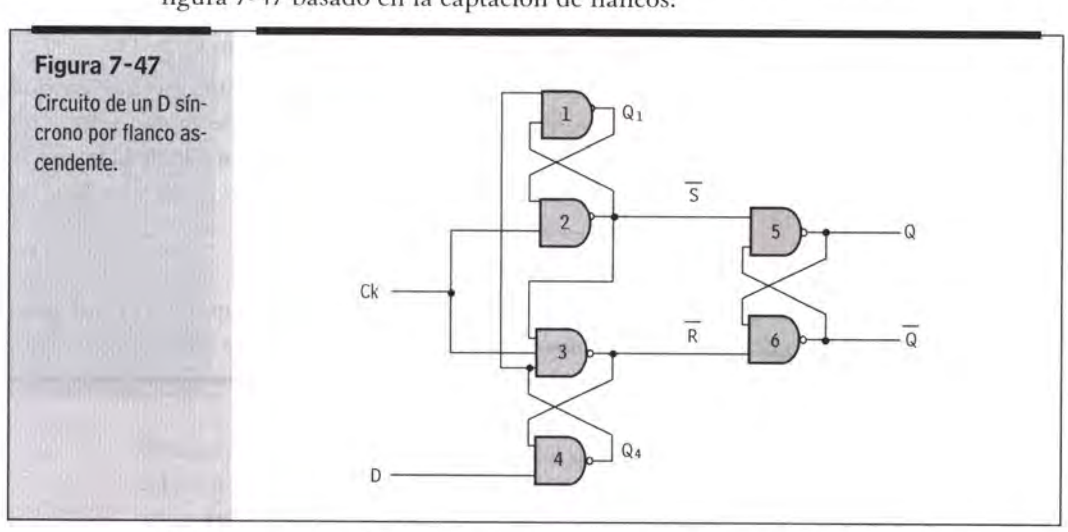


Figura 7-47
Circuito de un D síncrono por flanco ascendente.

En el esquema 7-47 anterior podemos ver tres biestables R-S. Dos en la entrada (1-2 y 3-4) con la misión de obtener las entradas $\overline{S}$ y $\overline{R}$ del biestable R-S de salida (5-6). Analicemos el circuito caso por caso:

1. Ck=0. Mantenimiento o memoria

(biestable no sincronizado)

Las puertas 2 y 3 ponen $\overline{S}=1$ y $\overline{R}=1$, y por tanto la salida Q no varía:

$$Q_t = Q_{t-1} \text{ y } \overline{Q_t} = \overline{Q_{t-1}}$$

2. Ck=↑ y D=0. Puesta a 0

(paso de Ck = 0 a Ck = 1)

Antes de producirse el flanco Ck = 0, luego $\overline{S} = \overline{R} = 1$, y como D = 0 entonces $Q_4 = 1$. Así, cuando llegue el flanco Ck = 1 las tres entradas de 3 estarán a 1 y su salida por tanto pasará a 0, es decir, $\overline{R} = 0$. Simultáneamente, las entradas $\overline{S}$ y Q_4 de 1 son 1, y por tanto $Q_1 = 0$ y por ello $\overline{S}$ será 1.

Si $\overline{S} = 1$ y $\overline{R} = 0$ entonces: Q = 0 y $\overline{Q} = 1$.

3. Ck = ↑ y D = 1. Puesta a 1

(paso de Ck = 0 a Ck = 1)

Antes del flanco, como Ck = 0, luego $\overline{R} = 1$, y por tanto $Q_4 = 0$, lo que a su vez conlleva que Q1 = 1. Cuando llegue el flanco, Ck = 1 y por tanto $\overline{S} = 0$.

Si $\overline{S} = 0$ y $\overline{R} = 1$ entonces: Q = 1 y $\overline{Q} = 0$.

4. Ck=1. Mantenimiento

(fin del flanco, pérdida de sincronismo)

La cuestión es: ¿si estando Ck=1 la entrada cambia, qué pasa en la salida?

Si D había sido 0 esto conllevaba $\overline{R}$ =0. Este valor hace que aunque cambie D, Q_4 no cambie, lo que asegura que no cambien ni $\overline{S}$ ni $\overline{R}$, y por tanto $Q_t=Q_{t-1}$.

Si D había sido 1 esto conllevaba que $\overline{S}$ =0 y $\overline{R}$ =1 y con ellas Q_4=0 y por tanto Q_1=1, lo que asegura junto con el Ck = 1 que $\overline{S}$ = 0, y por tanto $\overline{R}$ = 1. Es decir, aunque cambie D, no cambiarán ni $\overline{S}$ ni $\overline{R}$, y por tanto $Q_t = Q_{t-1}$.

Resumiendo, el valor de D sólo afecta al circuito en el primer instante en que Ck=1 -es decir en el flanco-, ya que posteriores cambios de D con Ck=1 no afectarán a la salida Q. Anotemos que esta báscula es idéntica a la de la figura 7-45 en cuanto a qué hace, pero no en cuanto a cómo lo hace.

7.7.5. Biestable T síncrono por flanco

El comportamiento del biestable T (figura 7-48) es muy sencillo, pero de gran utilidad en determinadas aplicaciones, como por ejemplo los contadores.

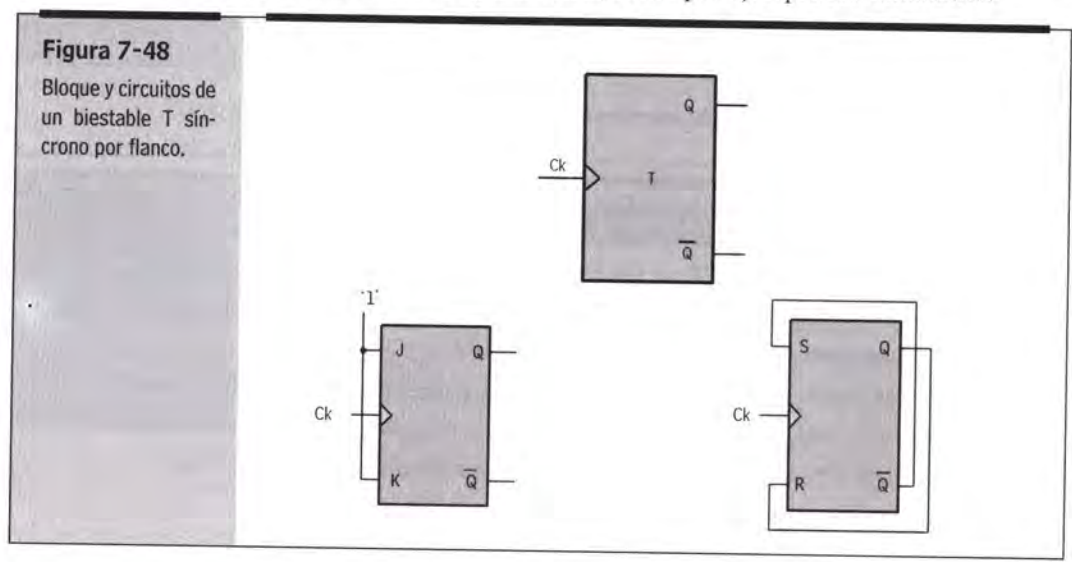


Figura 7-48

Bloque y circuitos de un biestable T síncrono por flanco.

A la vista del circuito 7-48, un biestable T siempre pasa al estado contrario del inmediatamente anterior. El cronograma de la figura 7-50 muestra este comportamiento y que la configuración en cascada de n biestables T (figura 7-49) es un divisor de frecuencia. En este caso:

$$f_{Q1} = \frac{f_{ck}}{2}; \qquad f_{Q2} = \frac{f_{Q1}}{2} = \frac{f_{ck}}{4}, \qquad f_{Q3} = \frac{f_{Q2}}{2} = \frac{f_{ck}}{8}.$$

Figura 7-49

Divisor de frecuencias.

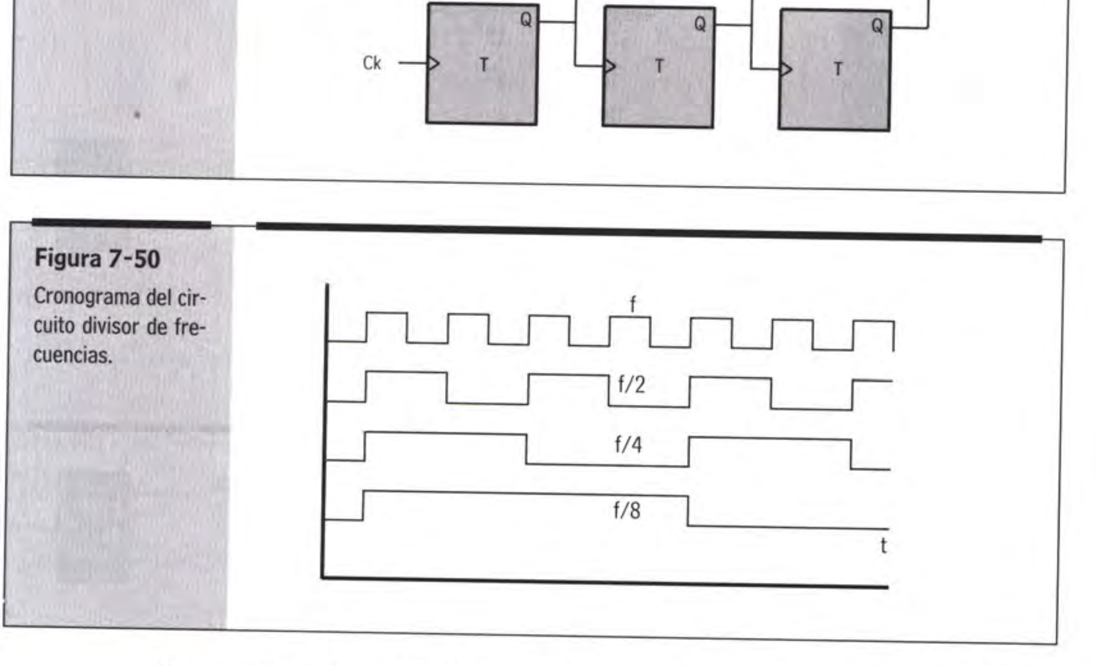


Figura 7-50

Cronograma del circuito divisor de frecuencias.

Una versión del biestable T es aquella que incluye una línea T que habilita o no el basculamiento del biestable. La figura 7-51 y la tabla de verdad 7-25 muestran este nuevo comportamiento.

Figura 7-51

Circuito del biestable T con entrada T.

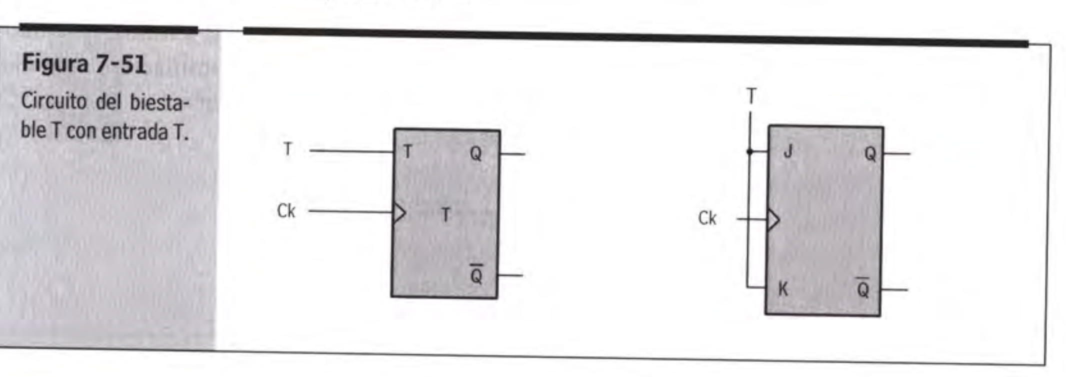


Tabla 7-25
Tabla de un biestable T.

Ck	T	Q_t
↑	1	$\overline{Q_{t-1}}$
↑	0	Q_{t-1}

Podemos obtener la ecuación booleana que representa a un biestable T a partir de la tabla de verdad 7-26.

Tabla 7-26
Tabla de verdad de un biestable T.

Ck	T	Q_{t-1}	Q_t
↑	1	0	1
↑	1	1	0
↑	0	0	0
↑	0	1	1

Del V-K, la ecuación resultante a implementar es:

$$Q_t = T \cdot \overline{Q_{t-1}} + \overline{T} \cdot Q_{t-1}, \text{ si } Ck = \uparrow$$

La tabla 7-27 es la tabla de excitación de un biestable T.

Tabla 7-27
Tabla de excitación de un T síncrono por flanco.

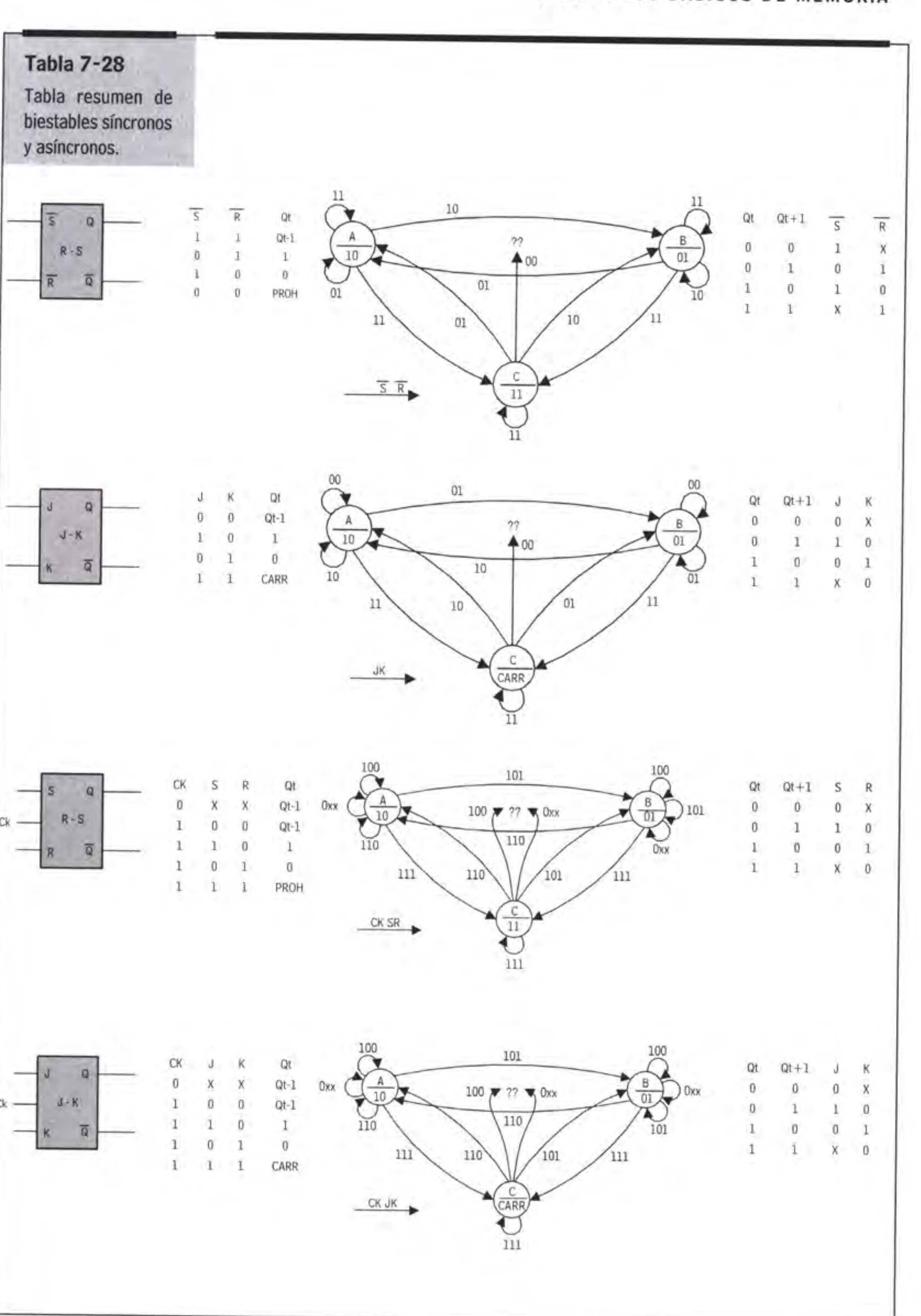
Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

7.8. Resumen de biestables

La tabla 7-28 resume el comportamiento de los biestables asíncronos, síncronos por nivel y síncronos por flanco. La tabla presenta su denominación, símbolo, tabla de verdad, diagrama de transición y su tabla de excitación.

Tabla 7-28

Tabla resumen de biestables síncronos y asíncronos.



$\overline{S}$	$\overline{R}$	Qt
1	1	Qt-1
0	1	1
1	0	0
0	0	PROH

Qt	Qt+1	$\overline{S}$	$\overline{R}$
0	0	1	X
0	1	0	1
1	0	1	0
1	1	X	1

J	K	Qt
0	0	Qt-1
1	0	1
0	1	0
1	1	CARR

Qt	Qt+1	J	K
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

CK	S	R	Qt
0	X	X	Qt-1
1	0	0	Qt-1
1	1	0	1
1	0	1	0
1	1	1	PROH

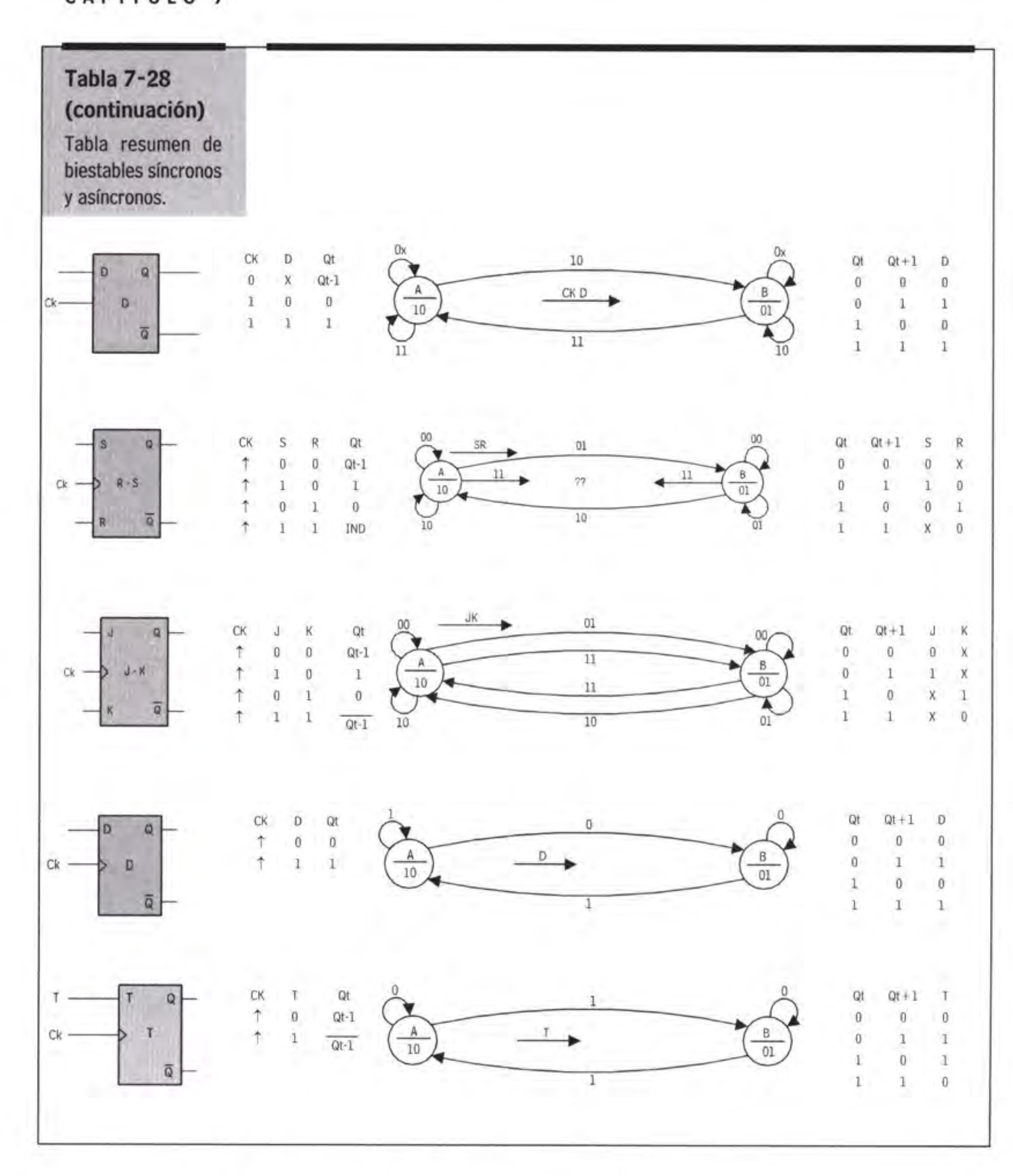
Qt	Qt+1	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

CK	J	K	Qt
0	X	X	Qt-1
1	0	0	Qt-1
1	1	0	1
1	0	1	0
1	1	1	CARR

Qt	Qt+1	J	K
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Tabla 7-28 (continuación)

Tabla resumen de biestables síncronos y asíncronos.



CK	D	Qt
0	X	Qt-1
1	0	0
1	1	1

Qt	Qt+1	D
0	0	0
0	1	1
1	0	0
1	1	1

CK	S	R	Qt
↑	0	0	Qt-1
↑	1	0	1
↑	0	1	0
↑	1	1	IND

Qt	Qt+1	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

CK	J	K	Qt
↑	0	0	Qt-1
↑	1	0	1
↑	0	1	0
↑	1	1	$\overline{Qt\text{-}1}$

Qt	Qt+1	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

CK	D	Qt
↑	0	0
↑	1	1

Qt	Qt+1	D
0	0	0
0	1	1
1	0	0
1	1	1

CK	T	Qt
↑	0	Qt-1
↑	1	$\overline{Qt\text{-}1}$

Qt	Qt+1	T
0	0	0
0	1	1
1	0	1
1	1	0

7.9. Conversión entre biestables

Una situación bastante común es hacer que un biestable se comporte como otro. Por ejemplo, supongamos que disponiendo de biestables tipo D, nuestro diseño contemple J-K (figura 7-52). La solución pasa por implementar el J-K con el D, utilizando para ello las ecuaciones booleanas características ya obtenidas y otras nuevas.

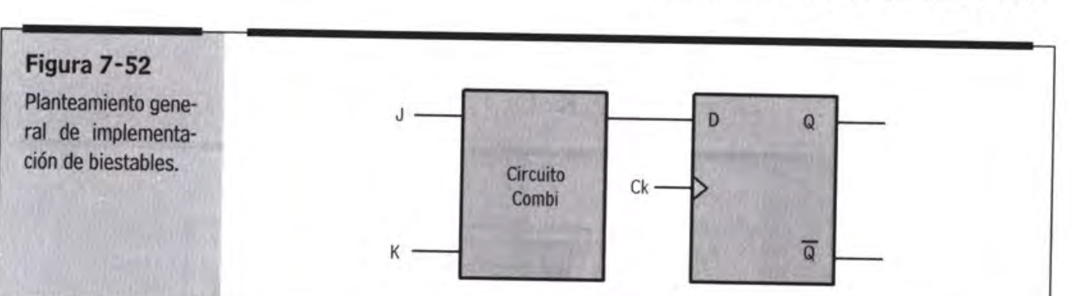


Figura 7-52
Planteamiento general de implementación de biestables.

Por ejemplo, implementar un J-K con D:

Del biestable J-K, de la tabla 7-20, tenemos que

$$Q_t = J \cdot \overline{Q_{t-1}} + \overline{K} \cdot Q_{t-1}$$

Del biestable D, de la tabla 7-23, a su vez que

$$Q_t = D$$

Juntando ambas ecuaciones resulta:

$$D = J \cdot \overline{Q_{t-1}} + \overline{K} \cdot Q_{t-1}$$

Implementando las anteriores ecuaciones tenemos el circuito de la figura 7-53, que se comporta en conjunto como un J-K síncrono por flanco.

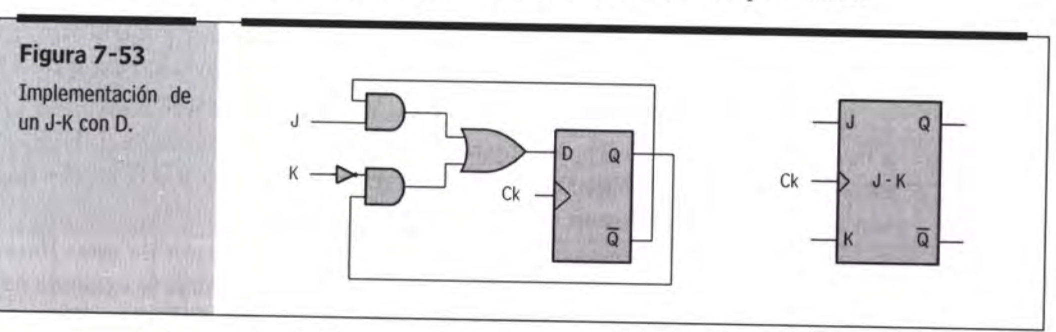


Figura 7-53
Implementación de un J-K con D.

Por ejemplo, implementar un J-K con un T. En este caso seguimos un método más sistemático: plantear la tabla de verdad 7-29, simplificarla e implementarla.

Tabla 7-29
T-V de un J-K basado en un T.

J	K	Q_{t-1}	Q_t	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1

Del V-K, la ecuación resultante a implementar es:

$$T = J \cdot \overline{Q_{t-1}} + K \cdot Q_{t-1}$$

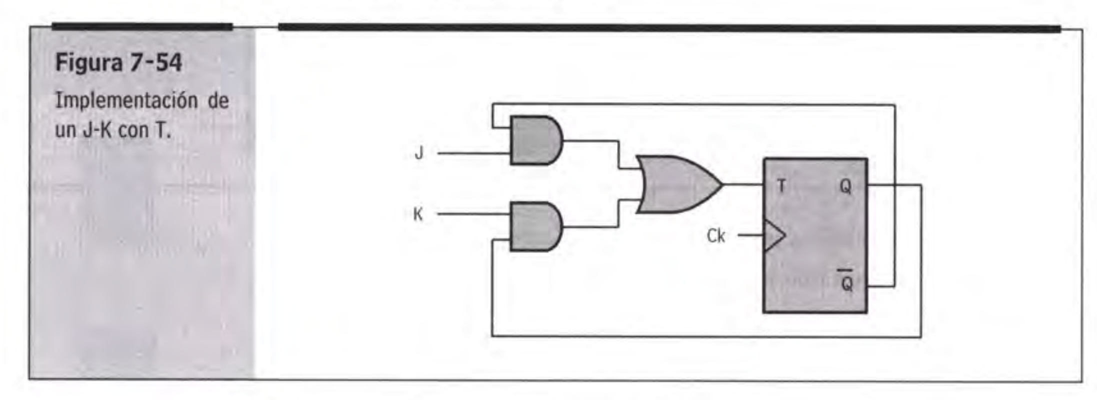


Figura 7-54
Implementación de un J-K con T.

7.10. Líneas asíncronas en un biestable

Las líneas asíncronas de un biestable cumplen una función muy importante, sobre todo cuando los biestables se integran en un diseño *real*. Su función es permitir al sistema forzar el valor de los biestables y con ellos el estado del sistema completo, por ejemplo parar un motor por emergencia, rearrancar un sistema bloqueado, etc. Estas líneas asíncronas suelen estar generalmente asociadas con situaciones de emergencia, problemas, bloqueo, errores, etc., que por tanto no son normalmente utilizadas, y que cuando lo son deben ser efectivas sin excepción.

La función de las líneas Preset y Clear -así se llaman- es forzar el biestable a 1 o a 0, respectivamente, independientemente del valor de las entradas -J, K, R, S, etc.- y del reloj. La principal diferencia de Preset y Clear con R o S, etc., es que son asíncronas; no necesitan del reloj, se lo *saltan*.

En un biestable R-S síncrono es muy fácil la implementación de estas líneas (figura 7-55), y una vez que están en este biestable, su presencia se extiende por el resto. Es decir, todos los biestables síncronos, ya sean J-K, R-S, D o T tienen líneas asíncronas de Puesta a 1 y Puesta a 0 (Preset y Clear) (figura 7-56).

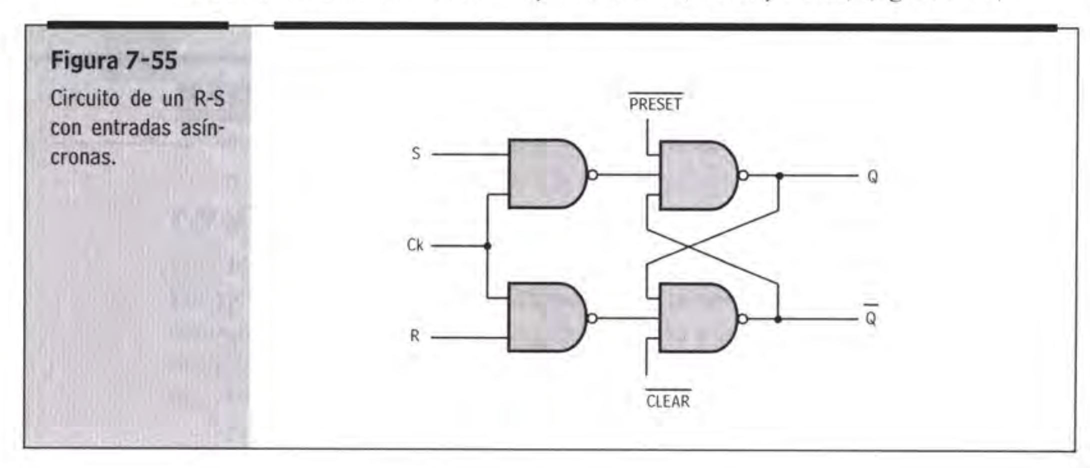
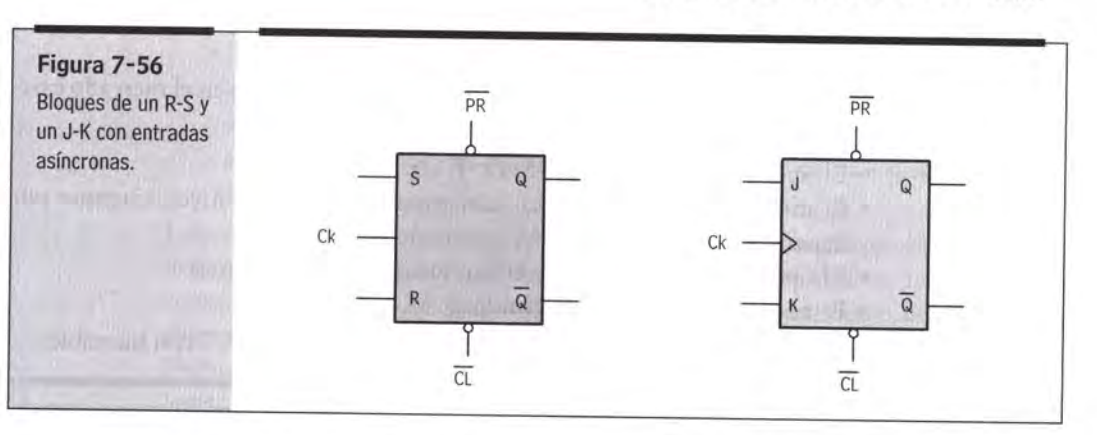


Figura 7-55
Circuito de un R-S con entradas asíncronas.

Figura 7-56
Bloques de un R-S y un J-K con entradas asíncronas.



La tabla 7-30 muestra el efecto de las líneas de Preset y Clear activas por nivel bajo -PR y CL- en un biestable R-S síncrono por nivel alto.

Tabla 7-30
Tabla de un R-S con entradas asíncronas.

Modo	Ck	$\overline{PR}$	$\overline{CL}$	S	R	Q_t
Asíncrono	X	0	1	X	X	1
	X	1	0	X	X	0
	1	1	1	0	0	Q_{t-1}
Síncrono	1	1	1	0	1	0
	1	1	1	1	0	1
	1	1	1	1	1	PROH
Memoria	X	1	1	X	X	Q_{t-1}
Prohibido	X	0	0	X	X	PROH

Ambas líneas $\overline{PR}$ y $\overline{CL}$ son activas por nivel bajo, como indican su barra de negación y el pequeño círculo en la correspondiente línea del biestable. La tabla 7-30 indica que nunca se deben activar ambas líneas de inicialización simultáneamente.

7.11. Circuitos MSI y aplicaciones de biestables

Los biestables son los elementos básicos de memoria, así que son imprescindibles en cualquier sistema secuencial. Ahora bien, su presencia destaca sobre todo en:

- Contadores. Sistema capaz de generar una determinada secuencia en un determinado código.
- Registros. Sistema capaz de almacenar información de distintas maneras y también capaz de ofrecerla de distintas maneras.
- Autómata. Controla la evolución de un sistema en base a sus entradas y estados.

En estas aplicaciones se centrarán los siguientes capítulos.

Respecto a los circuitos integrados con biestables disponibles, en el mercado existe una gran variedad según:

- El tipo de biestable: J-K, R-S, D T, etc.
- El modo de funcionamiento: asíncrono, síncrono por nivel, síncrono por flanco, M/S, etc.
- El tipo de líneas de Preset y Clear: asíncronas y síncronas.
- El número de biestables incluidos.

La tabla 7-31 intenta resumir las características de distintos CI con biestables.

Tabla 7-31
Descripción de circuitos MSI con biestables.

Dispositivo	Biestable	Sincronismo	Preset	Clear	Número	Nivel activo
74279 (1)	R-S	Asíncrono	No	No	4	bajo
7475 (2)	LATCH	Nivel alto	No	No	4	alto
74116 (3)	D	Nivel alto	No	$\overline{CL}$	2	alto
7474	D	Flanco ↑	$\overline{PR}$ Asín.	$\overline{CL}$ Asín.	2	alto
7473	J-K	Flanco ↓	No	$\overline{CL}$ Asín.	2	alto
7476	J-K	Flanco ↓	$\overline{PR}$ Asín.	$\overline{CL}$ Asín.	2	alto
74108 (4)	J-K	Flanco ↓	$\overline{PR}$ Asín.	$\overline{CL}$ Común	2	alto
74111 (5)	J-K	M/S ⊓	$\overline{PR}$ Asín.	$\overline{CL}$ Asín.	2	alto

(1) En dos biestables sus entradas $\overline{CL}$ son el producto de dos entradas S1 y S2.

(2) Los dos primeros cerrojos comparten la línea ENABLE; igual para los dos últimos. Su implementación no está libre de riesgos.

(3) La línea ENABLE es el producto de dos entradas. El biestable D está libre de riesgos.

(4) El reloj es común a todos los biestables.

(5) Es un biestable J-K M/S con cierre de datos.

$\overline{CL}$ común significa que sólo hay una línea CL, común a todos los biestables.

$\overline{CL}$ o $\overline{PR}$ Asín. significa que dicha línea de inicialización es asíncrona.

7.12. Parámetros tecnológicos y temporales en un biestable

En los anteriores apartados hemos mostrado la lógica que siguen los distintos biestables. En este apartado abordaremos las restricciones temporales impuestas por cada biestable para asegurar un funcionamiento válido.

Las restricciones vienen dadas por dos aspectos tecnológicos comunes a todos los biestables:

- La línea de reloj que aporta el sincronismo a los biestables es considerada cuadrada, con flancos verticales. En la realidad esto no es así, lo que complica la evolución del biestable.

- Las puertas de un biestable están retroalimentadas, afectándose unas a otras. Así, las entradas no deberán cambiar mientras se produzcan las realimentaciones que hayan generado. Dichas realimentaciones hacen que el biestable pase por estados transitorios, en los que no es aconsejable variar las entradas.

Las consideraciones anteriores respecto del reloj y de la realimentación imponen determinadas restricciones al uso de los biestables. Éstas deben ser tenidas en cuenta por el diseñador para asegurar un buen funcionamiento del sistema. A continuación describimos los principales parámetros y restricciones temporales. Dichos parámetros están en continua mejora gracias a la evolución de las tecnologías semiconductoras.

Tiempo de propagación o retardo del biestable (delay time). t_{pd}, t_{pLH} y t_{pHL}.

Es el tiempo necesario para que el efecto de un cambio en la entrada se haga estable en la salida.

Son distintos tiempos según la salida tenga que pasar de alto a bajo -t_{pHL}-, o viceversa -t_{pLH}-. El valor t_{pd} es el valor medio de los dos anteriores.

Tiempo de establecimiento (setup time). t_s

Es el tiempo mínimo anterior al flanco de disparo en que las entradas no deben variar. Es decir, es el tiempo que necesita el biestable para asentar las entradas antes de que llegue el flanco de disparo.

Tiempo de mantenimiento (hold time). T_h

Es el tiempo mínimo posterior al flanco de disparo en que las entradas no deben variar. Es decir, es el tiempo que necesita el biestable para procesar las entradas. Los valores de t_H y t_S no suelen ser iguales.

Anchura del reloj. t_{WH} y t_{WL}

Es el tiempo mínimo que los biestables exigen que duren los niveles alto y bajo del reloj, respectivamente t_{WH} y t_{WL}.

Frecuencia máxima. f_{max}

Es la máxima frecuencia permitida al reloj del biestable. Superada ésta el comportamiento del biestable puede ser erróneo.

Tiempo de preset y clear.

Es el tiempo mínimo que debe durar el nivel activo de las líneas asíncronas de puesta a 1 y a 0 para que fuercen el valor del biestable. Su valor suele estar incluido en t_{pHL} y t_{pLH}.

Los anteriores parámetros deben ser conocidos y respetados por el diseñador. No es intención de este libro entrar en este tipo de pormenores, pero piénsese en que el reloj y las entradas no tienen por qué estar sincronizadas, y así puede que las entradas cambien en cualquier momento (por ejemplo, un pulsador) sin respetar los tiempos de mantenimiento y/o asentamiento, con el consiguiente mal funcionamiento. Estas situaciones pueden diversificarse y complicarse, dando lugar al problema conocido como metaestabilidad (discutido en el capítulo 10).

En la tabla 7-32 se muestran los valores de los parámetros para los CI básicos 7473, 7474, 7475 y 7476. Para todos ellos se dan los valores correspondientes a dos tecnologías: HC (High Speed) y LS (Low Power Schottky); la primera es más moderna y rápida.

Tabla 7-32

Parámetros temporales de CI con biestables.

	74HC73	74LS73	74HC74	74LS74	74HC75	74LS75	74HC76	74HCT76
fmax (MHz)	60	15	50	25	-- (1)	-- (1)	55	15
tpHL (ns)	20	40	20	40	8	9	18	40
tpLH (ns)	20	25	20	25	8	15	18	25
ts (ns)	9	20	9	20	1	20	6	20
th (ns)	0	0	0	5	0	5	0	0
tWH (ns)	8	20	8	30	8	20	8	20
tWL (ns)	8	47	8	37	8	20	8	47

(1) Al ser un latch no tiene sentido hablar de Ck.

7.13. Resumen

Antes de estudiar los sistemas combinacionales hicimos lo propio con las puertas lógicas, resultando que de la combinación de éstas se obtenían sistemas combinacionales de creciente complejidad.

Del mismo modo, el análisis y diseño de los sistemas secuenciales pasa por el estudio previo de los biestables. Los biestables son a los secuenciales lo que las puertas a los combinacionales; eso sí, son algo más complejos.

Para poder seguir los temas siguientes, al acabar este capítulo el lector debe conocer al menos qué hace cada biestable, resultando también interesante que sepa el porqué y el cómo.



CAPÍTULO

8

REGISTROS

8.1. Introducción

Los registros son uno de los dispositivos digitales más comunes e importantes en el diseño digital, y sin embargo su análisis y diseño es muy sencillo.

Un registro es un sistema secuencial síncrono capaz de almacenar información durante un tiempo. Un registro almacena n bits, estando cada bit almacenado en un biestable.

Si desde el punto de vista funcional un registro almacena información, desde el punto de vista de la implementación un registro no es más que una sencilla conexión entre biestables. De hecho, los biestables fueron desarrollados teniendo en cuenta su uso en los registros.

En la figura 8-1 adjunta vemos cómo usar las básculas D y R-S en registros; el uso de J-K no es común.

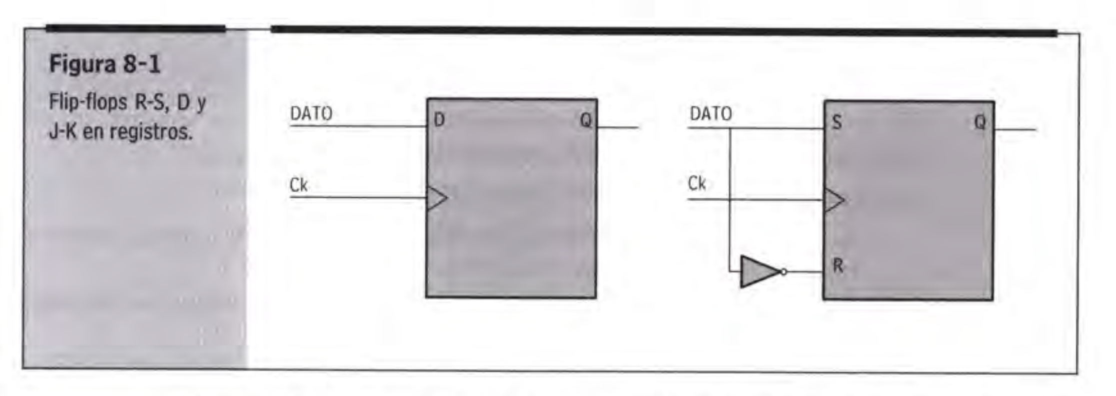


Figura 8-1
Flip-flops R-S, D y J-K en registros.

Observando las básculas es claro que el D o R-S síncrono por flanco se carga con el valor de DATO cuando recibe un flanco ascendente en el reloj; una vez desaparecido el flanco, el biestable no cambia su contenido y memoriza el bit hasta la llegada del siguiente flanco. Si el biestable es síncrono por nivel, entonces la salida del biestable es igual a DATO si el reloj está a nivel alto, y cuando el reloj pasa a nivel bajo el biestable memoriza el último valor de DATO.

Los biestables suelen ser D o R-S. El tipo D suele ser síncrono por flanco o por nivel, mientras que el R-S es del tipo Maestro-Esclavo.

Desde el punto de vista de la descripción funcional en un registro la entrada y la salida pueden ser paralelo o serie. Así, los registros pueden ser (ver figura 8-2):

- Registro con entrada paralelo y salida paralelo.
- Registro con entrada paralelo y salida serie.
- Registro con entrada serie y salida paralelo.
- Registro con entrada serie y salida serie.

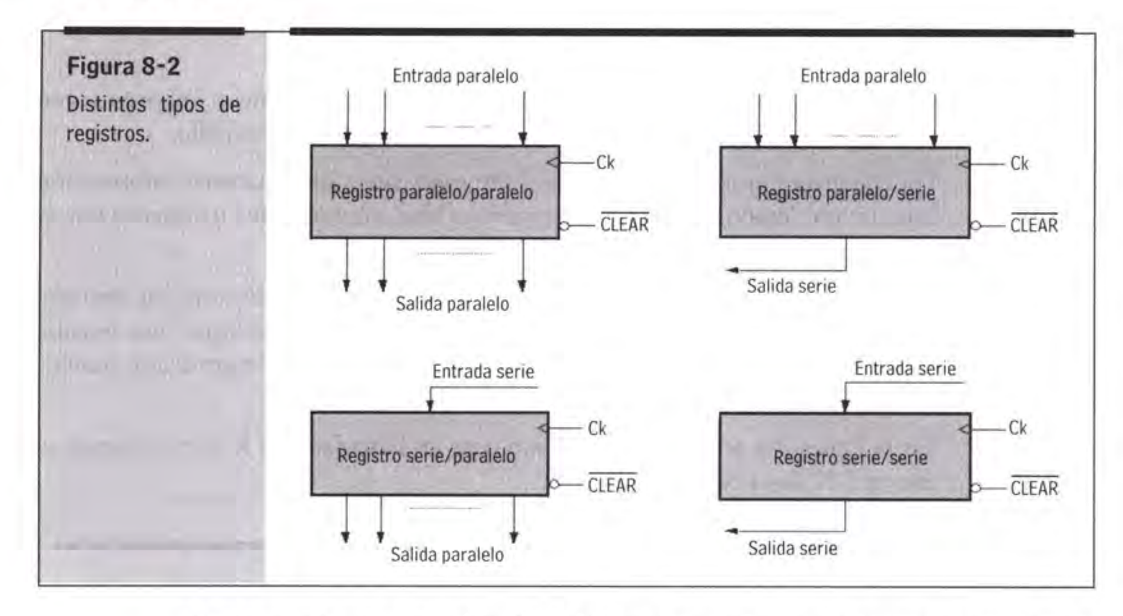


Figura 8-2
Distintos tipos de registros.

En los siguientes apartados describiremos someramente dichos registros, implementándolos con los biestables vistos en el capítulo anterior.

Las líneas más comunes en un registro son:

- Entrada paralelo. Cada biestable recibe su propio bit de entrada. Los biestables actúan en paralelo.
- Salida paralelo. Cada biestable ofrece su propio bit de salida. Los biestables actúan en paralelo.
- Entrada serie. La entrada se produce bit a bit. Los biestables actúan secuencialmente.
- Salida serie. La salida se produce bit a bit. Los biestables actúan secuencialmente.
- Reloj. Indica cuándo se produce la carga del registro. Marca el ritmo de la carga o descarga secuencial.
- Clear. Línea generalmente asíncrona cuya activación provoca que todo el registro se cargue con ceros; que se inicialice.
- Preset. Idéntica a la anterior, pero con unos. Esta línea no suele aparecer.
- Inhibición de Reloj. Para que el reloj actúe con normalidad, y con él el registro, es necesario que la línea Inhibición de Reloj esté inactiva. Si se activa la señal, el reloj queda anulado y por tanto el registro queda también inhabilitado.

En cuanto a las aplicaciones de los registros, son variadas, pero en primer lugar atendemos a esta clasificación:

- Registros de almacenamiento: paralelo-paralelo y serie-serie.
- Conversión de datos: paralelo-serie y serie-paralelo.

La aplicación de los registros de almacenamiento es simplemente almacenar información; ésta llega en paralelo en un instante y se ofrece en paralelo hasta una nueva carga (o se recibe y entrega en serie).

Sin embargo, en los paralelo/serie o serie/paralelo hay una conversión. La entrada es en paralelo y la salida se ofrece en serie, o viceversa. Esta conversión adapta la forma en que llegan los datos de entrada y la forma en la que van a ser procesados por el sistema, es decir, los conversores son previos al proceso (codificadores, sumadores, etc.); sólo ordenan la entrada para ser procesada (ver figura 8-3).

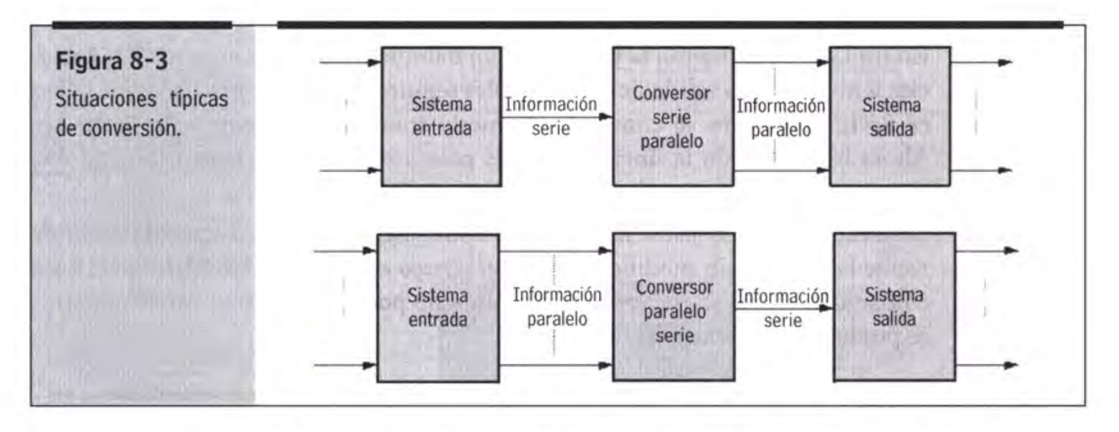


Figura 8-3
Situaciones típicas de conversión.

Aunque el almacenamiento de información y la conversión son las principales aplicaciones de los registros, éstos también tienen otras:

- Contadores basados en códigos especiales.
- Acumuladores en sumadores serie.
- Almacenamiento en microprocesadores en forma de BUS.
- Almacenamiento auxiliar en unidades de entrada/salida.

8.2. Registro paralelo/paralelo

En este caso, la presencia de un flanco ascendente hace que se cargue el registro con el contenido de las líneas de entrada. El valor cargado se mantiene en las salidas de los biestables mientras no se produzca otro flanco ascendente.

El circuito de la figura 8-4 muestra que un registro es una simple acumulación de flip-flop's.

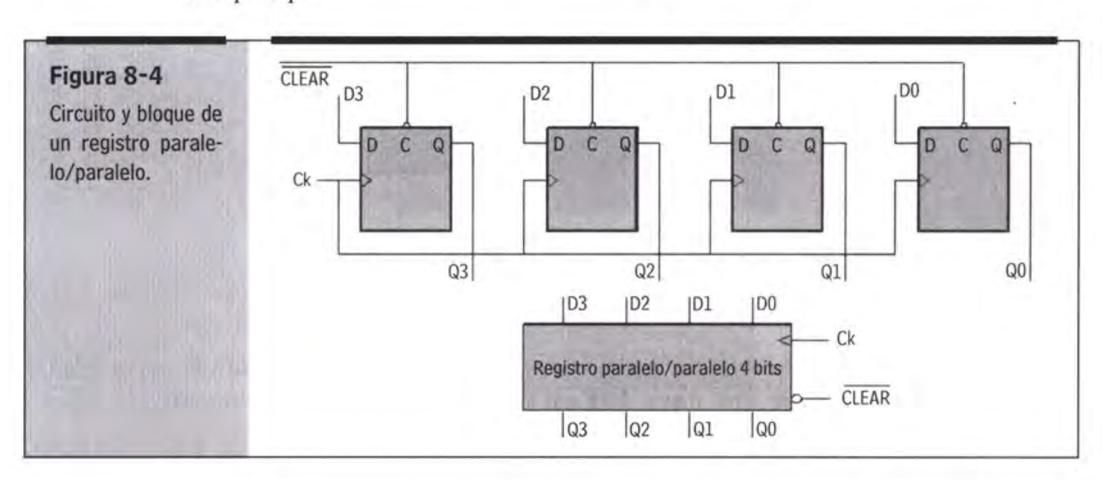


Figura 8-4
Circuito y bloque de un registro paralelo/paralelo.

El registro de la figura 8-4, basado en básculas D activas por flanco ascendente, dispone de una línea de puesta a cero o inicialización asíncrona activa por nivel bajo.

Una versión muy utilizada del anterior circuito consiste en modificar el sincronismo, que pasa de ser activo por flanco a serlo por nivel. El registro así diseñado se denomina LATCH.

En un LATCH, cuando la línea del reloj habitualmente denominada ENABLE esté a nivel alto, la salida de los biestables seguirá las variaciones de las entradas, es decir, el registro se comporta de modo transparente frente a las entradas. Ahora bien, cuando la línea ENABLE pase a nivel bajo el registro retendrá el último valor de las entradas antes del flanco descendente.

Resumiendo, si el registro se encuentra entre la entrada y un sistema, este último recibe la entrada sin modificaciones *si el cerrojo está abierto* -ENABLE = 1-; sin embargo, *si el cerrojo está cerrado* el sistema no podrá apreciar las variaciones que se produzcan a la entrada.

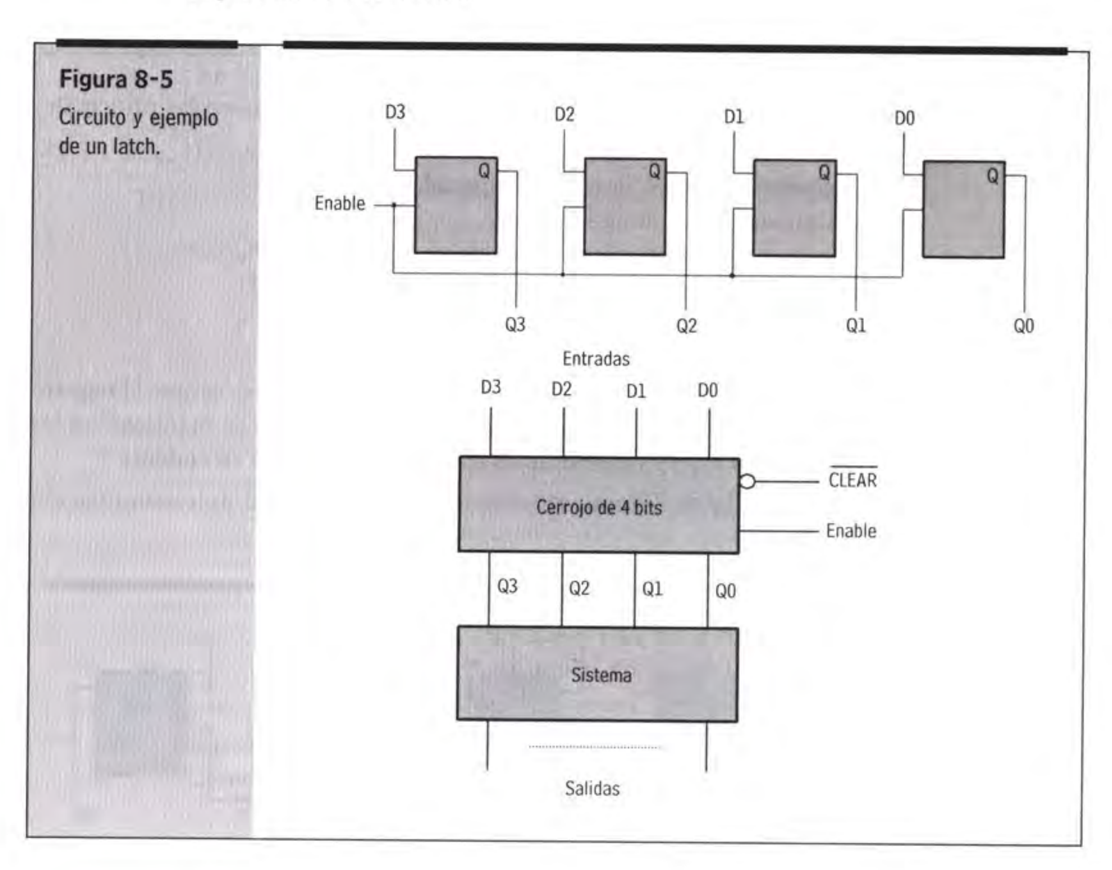


Figura 8-5
Circuito y ejemplo de un latch.

En el caso del registro cerrojo es típico que la salida esté controlada por triestados mediante una línea de Enable Output. La figura 8-6 muestra el registro 74373 paralelo-paralelo de 8 bits.

Figura 8-6
Registro paralelo/paralelo con Enable Output.

8.3. Registro serie/serie: registro de desplazamiento

En un registro serie-serie la entrada es bit a bit, al igual que la salida. Que la entrada sea serie supone que los bits llegan uno a uno, así, una entrada de cuatro bits llega en cuatro veces. Del mismo modo, la salida en serie de cuatro bits supone que éstos se presentan de uno en uno, en cuatro veces. Claramente, paralelo y serie son antitéticos; el primero supone inmediatez y simultaneidad, mientras que el segundo supone secuencialidad y orden.

El circuito de la figura 8-7 es un registro de desplazamiento a la derecha de cuatro bits con puesta a cero asíncrona. Se le denomina de desplazamiento, ya que al presentarse los bits en serie éstos se desplazan a través del registro.

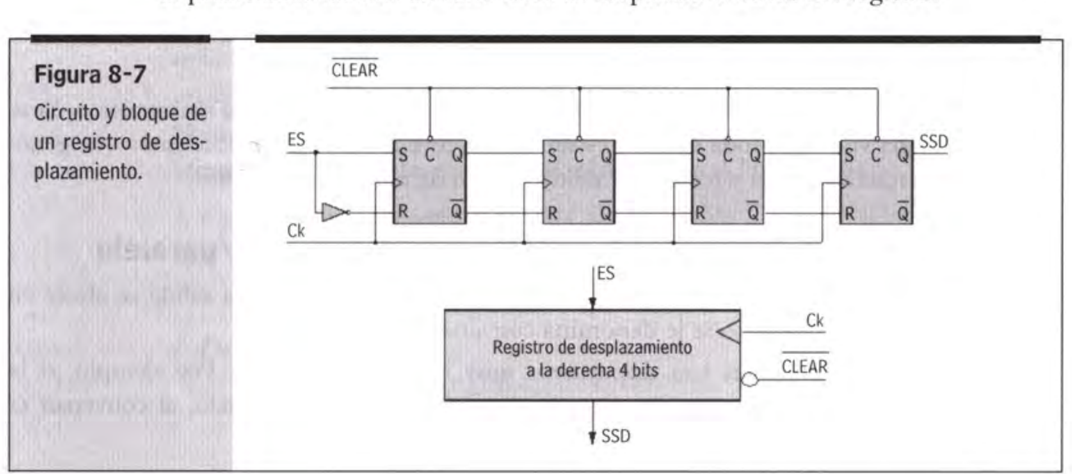


Figura 8-7
Circuito y bloque de un registro de desplazamiento.

Un registro paralelo-paralelo almacenaba el valor de las entradas -se cargaba- en un flanco de reloj. Ahora la pregunta es: ¿cuántos flancos necesita el registro anterior para cargarse con el valor de los cuatro bits de entrada? La respuesta es cuatro flancos, los mismos que hacen falta para descargar el registro.

Por ejemplo, veamos cómo se carga el registro con la secuencia 1011 (figura 8-8). El primer valor es un 1 que al llegar al flanco ascendente pasa a ser almacenado en Q_3, al llegar al segundo flanco en la entrada hay un 0 y por tanto el Q_3 se cargará con un 0, pero simultáneamente a esta carga el Q_2 tomará el valor que le entregue Q_3, es decir, un 1. El proceso se repite para los dos siguientes unos.

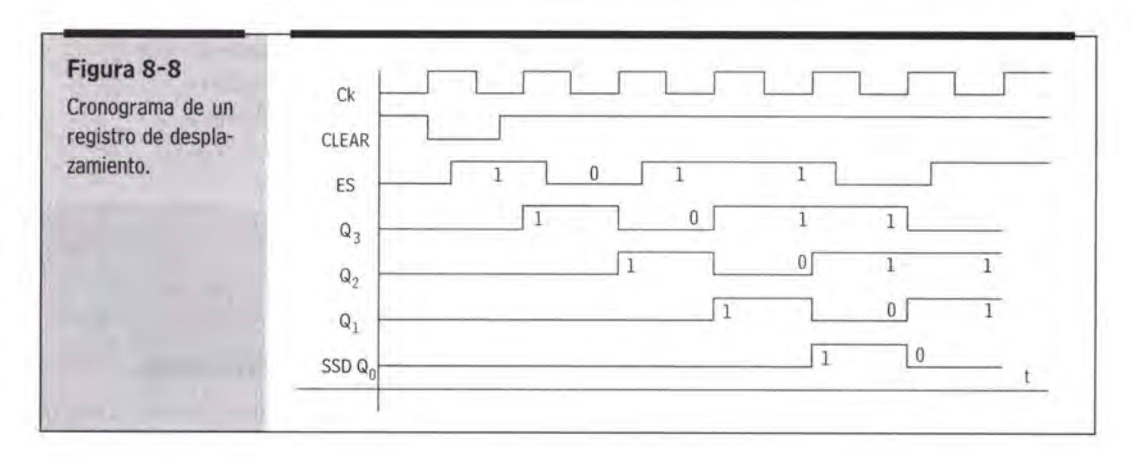


Figura 8-8
Cronograma de un registro de desplazamiento.

Una pregunta que queda por responder es: ¿al cargarse Q_3 con un 0 no obliga a Q_2 a cargarse con el mismo valor? La respuesta es no, pues cuando Q_3 toma el valor 0 ya no está presente el flanco, y por tanto su efecto no se notará hasta el siguiente flanco, lo que asegura un buen funcionamiento.

En el cronograma de la figura 8-8 vemos cómo el primer 1 pasados 4 flancos está en la salida SSD, y cómo este 1 alcanza la salida desplazándose a través de las básculas. El cronograma muestra claramente *el ritmo* del desplazamiento. También resulta claro cómo toda entrada de datos supone una salida de los anteriores, y viceversa.

Un registro de desplazamiento es muy exigente en cuanto al sincronismo. Ya se ha visto que toda entrada es salida, y viceversa, así pues los sistemas que generen la entrada y reciban la salida habrán de estar sincronizados.

8.4. Registro serie/paralelo: conversor serie/paralelo

En este caso la entrada se produce bit a bit, mientras que la salida se ofrece en paralelo, por eso se le denomina conversor serie-paralelo.

Los conversores son dispositivos muy usuales y sencillos. Por ejemplo, si la entrada se entrega en serie pero se va a procesar en paralelo, el conversor es imprescindible.

BIBLIOTECA CADIZ PUBLICA

Como vemos, el circuito de la figura 8-9 es idéntico al anterior, sólo cambia la disponibilidad de las salidas. De hecho, el anterior registro serie-serie es un registro serie-paralelo donde sólo se usa una de las salidas. Cabe insistir en que aunque ambos circuitos son iguales, su funcionalidad y condiciones de uso son totalmente distintas.

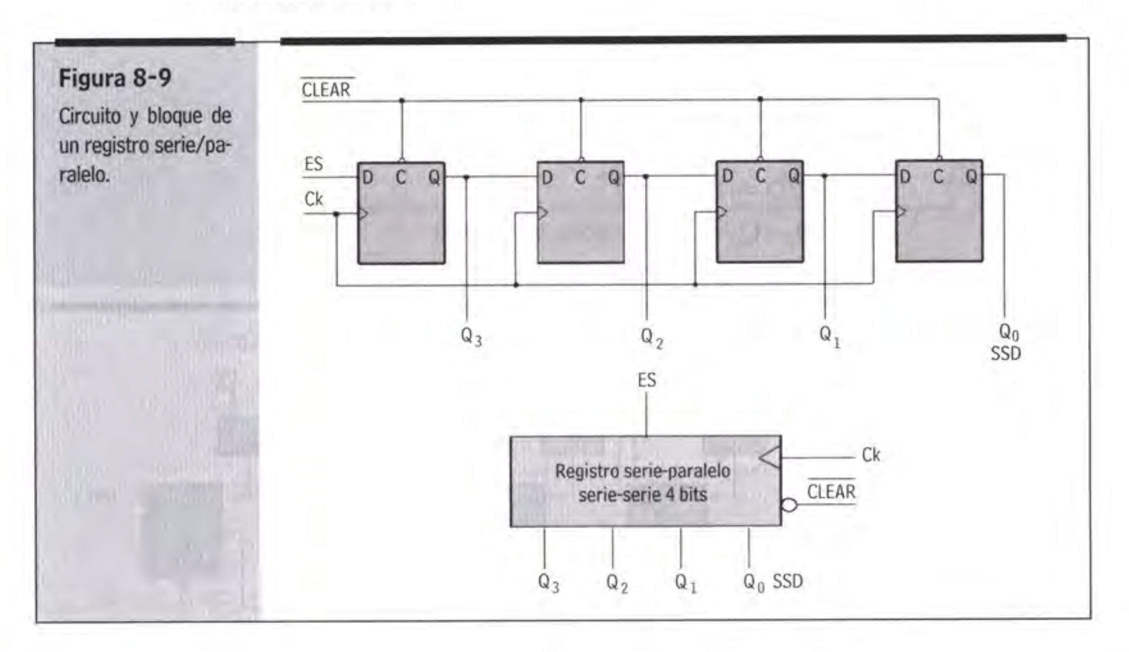


Figura 8-9
Circuito y bloque de un registro serie/paralelo.

8.5. Registro paralelo/serie: conversor paralelo/serie

En este conversor el registro se carga en paralelo en un solo flanco, mientras que la salida se ofrece bit a bit, en serie. Así, n bits se cargan en un flanco y se descargan en n bits más.

El circuito de la figura 8-10 se complica respecto de los anteriores, ya que las entradas de las básculas son controladas por un Mx que decide si la báscula recibe la entrada en paralelo o de la anterior báscula en desplazamiento. Lo dicho se controla mediante una línea MODO, con un 0 se produce la carga en paralelo y con un 1 el desplazamiento, o viceversa; la línea se conoce como $\overline{\text{CARGA}}$/DESPL.

Veamos un ejemplo en el que primero se carga el registro con 1101 y luego se ofrece su contenido en serie, desplazándolo hacia la derecha.

1. Carga síncrona del registro

$\overline{C}/D = 0 \quad D_3 - 0 = 1101 \quad Ck = \uparrow$

$Q_3 - Q_0 = 1101$ y $SSD = Q_3 = 1$

2. Desplazamiento

$\overline{C}/D = 1 \quad Ck = \uparrow$

$Q_3 - Q_0 = 0110$ y $SSD = Q_3 = 0$

3. Desplazamiento

$\overline{C}/D = 1 \quad Ck = \uparrow$

$Q_3 - Q_0 = 0011$ y $SSD = Q_3 = 1$

4. Desplazamiento

$\overline{C}/D = 1 \quad Ck = \uparrow$

$Q_3 - Q_0 = 0001$ y $SSD = Q_3 = 1$

5. Final de la descarga

$\overline{C}/D = 1 \quad Ck = \uparrow$

$Q_3 - Q_0 = 0000$ y $SSD = Q_3 = 0$

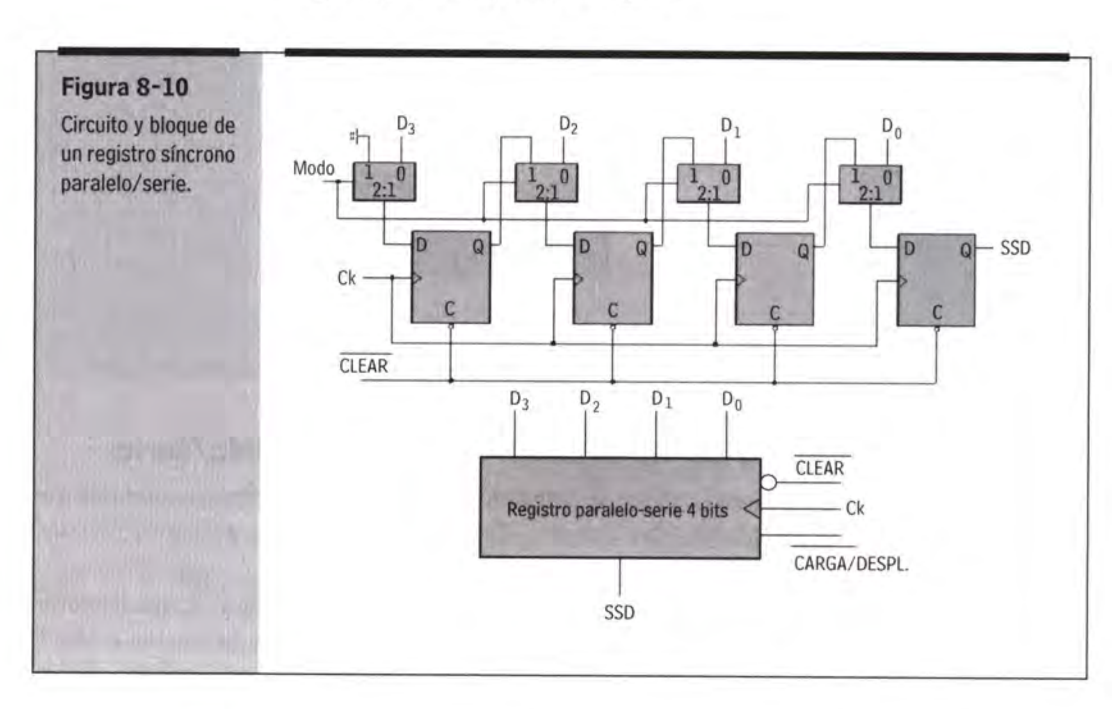


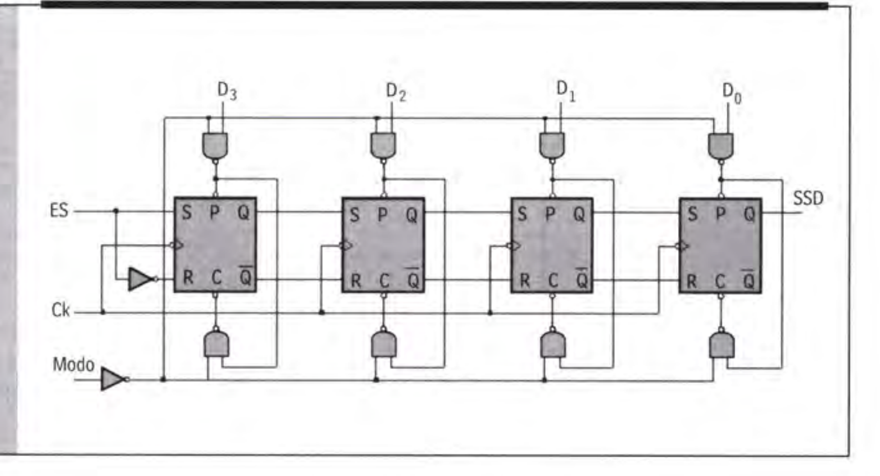
Figura 8-10
Circuito y bloque de un registro síncrono paralelo/serie.

En el circuito anterior resulta destacable que la línea 1 del primer Mx está fijada a 0. En realidad podría estar a cualquier valor, pues el funcionamiento es indiferente a él. En el registro 8-10, pasados los 4 flancos de descarga el registro queda cargado con ceros.

En el circuito anterior, tanto el desplazamiento como la carga en paralelo del registro se dan síncronamente, es decir, el registro está activo sólo si hay flancos en el reloj. En el circuito de la figura 8-11 el desplazamiento sigue siendo síncrono, como es obligado, pero la carga en paralelo es asíncrona, es decir, basta que la línea de MODO sea 0, sin necesidad de flancos. Este asincronismo se consigue utilizando las líneas asíncronas de Preset y Clear.

Figura 8-11

Circuito y bloque de un registro asíncrono paralelo/serie.



Veamos un ejemplo, si $D_3D_2D_1D_0 = 1011$ y MODO = 0, entonces

- $\overline{P_3} = \overline{D_3 \cdot \overline{MODO}} = \overline{1 \cdot 1} = 0$

  $\overline{C_3} = \overline{P_3 \cdot \overline{MODO}} = \overline{0 \cdot 1} = 1$

  y por tanto $Q_3 = 1$

- $\overline{P_2} = \overline{D_2 \cdot \overline{MODO}} = \overline{0 \cdot 1} = 1$

  $\overline{C_2} = \overline{P_2 \cdot \overline{MODO}} = \overline{1 \cdot 1} = 0$

  y por tanto $Q_2 = 0$

- $\overline{P_1} = \overline{D_1 \cdot \overline{MODO}} = \overline{1 \cdot 1} = 0$

  $\overline{C_1} = \overline{P_1 \cdot \overline{MODO}} = \overline{0 \cdot 1} = 1$

  y por tanto $Q_1 = 1$

- $\overline{P_0} = \overline{D_0 \cdot \overline{MODO}} = \overline{1 \cdot 1} = 0$

  $\overline{C_0} = \overline{P_0 \cdot \overline{MODO}} = \overline{0 \cdot 1} = 1$

  y por tanto $Q_0 = 1$

Vemos que la carga paralelo asíncrona se produce.

El circuito en su conjunto tiene una desventaja: no dispone de línea CLEAR de inicialización. Esto es así porque el circuito utiliza las líneas de Preset y Clear como líneas normales, despojándolas de su condición de externas o de emergencia. Esta ausencia de línea de inicialización podría ser suplida modificando el circuito.

En los cronogramas de la figura 8-12 se clarifica la diferencia entre un circuito y otro, que si bien es muy sencilla de entender, desde un punto de vista funcional es de gran importancia, como corresponde a los conceptos de síncrono y asíncrono.

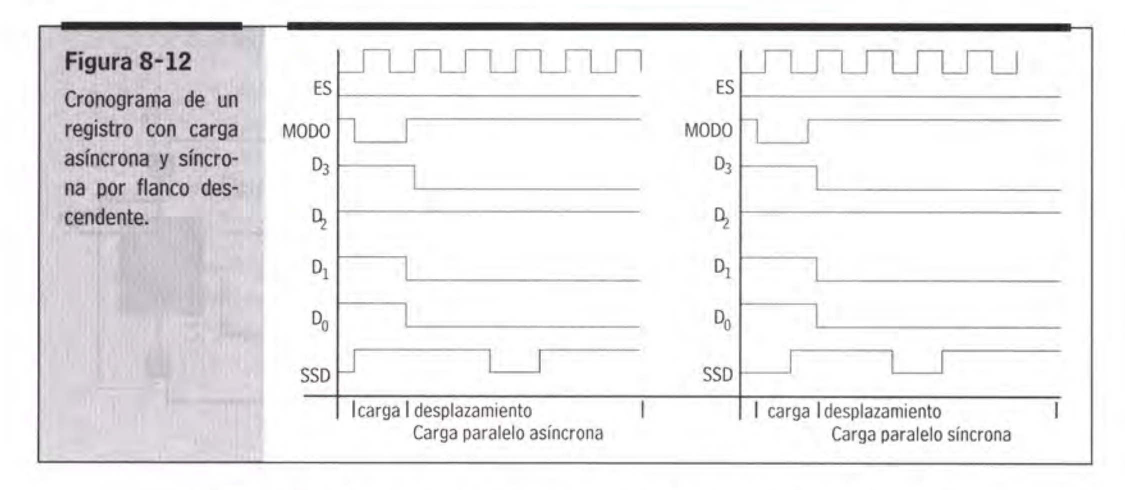


Figura 8-12

Cronograma de un registro con carga asíncrona y síncrona por flanco descendente.

8.6. Registro de desplazamiento derecha/izquierda

En este registro su contenido puede desplazarse hacia la derecha o la izquierda, según el valor de una línea MODO. Si MODO es 0 el desplazamiento es a la derecha; y lo contrario, si MODO es 1 el desplazamiento es a la izquierda, o viceversa. Según lo anterior, la línea suele llamarse DERECHA/$\overline{\text{IZQUIERDA}}$.

La implementación utiliza Mx para dirigir el desplazamiento hacia la derecha o la izquierda. El circuito resultante es el de la figura 8-13.

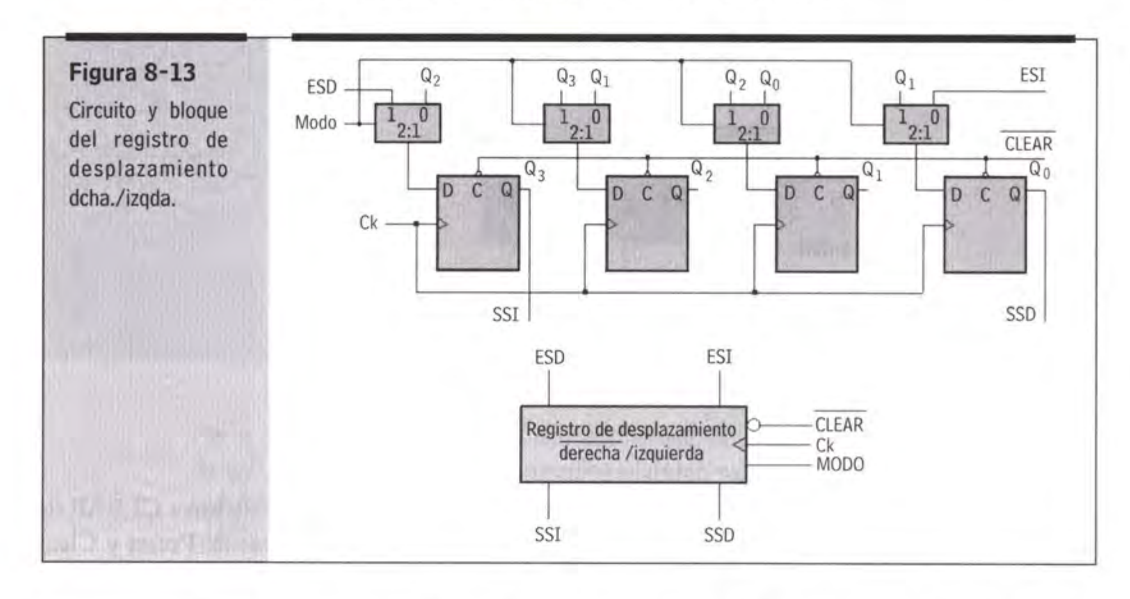


Figura 8-13

Circuito y bloque del registro de desplazamiento dcha./izqda.

En este circuito es destacable la denominación de las líneas de entrada serie, ESD y ESI. La extrañeza aparece al observar que ESD -entrada serie derecha- está a la izquierda, y viceversa para ESI. La razón no es otra que buscar la coherencia con el nombre de las salidas SSD y SSI, aunque siempre se pueden cambiar los nombres.

8.7. Registro universal

Este registro reúne las características de los anteriormente diseñados y explicados:

- Desplazamiento a la derecha.
- Desplazamiento a la izquierda.
- Carga en paralelo.
- Inhibición o no operación.

Si bien los tres primeros son claros, el último es novedoso. Cuando un registro está inhibido resulta que aunque en el reloj haya flancos el registro no variará su contenido, es decir, lo memorizará. Esta función es de gran importancia cuando varios registros interactúan.

El circuito de la figura 8-14 utiliza Mx, en este caso 4:1. Los distintos modos de funcionamiento son elegidos mediante las líneas de selección S_1 y S_0, resultando la tabla 8-1.

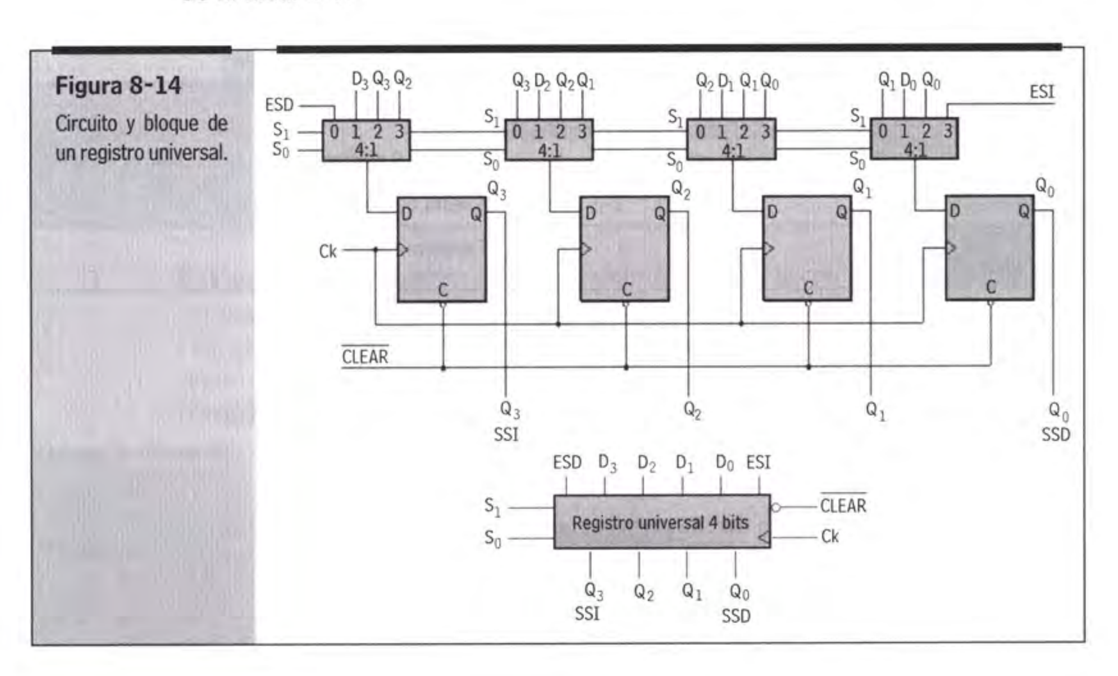


Figura 8-14
Circuito y bloque de un registro universal.

Tabla 8-1
Registro universal.

S_1	S_0	Función
0	0	Desplazamiento Derecha
0	1	Carga paralelo
1	0	Inhibición
1	1	Desplazamiento Izquierda

Siguiendo una estructura como la anterior se pueden diseñar registros con distintas funciones, sin más que cambiar las líneas de selección de los multiplexores. Por ejemplo, el circuito de la figura 8-15 representa un registro con la funcionalidad de la tabla 8-2.

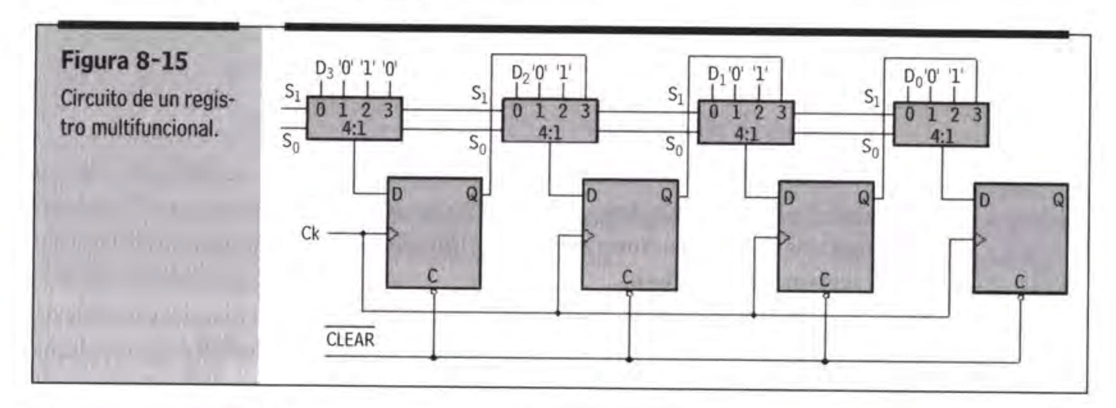


Figura 8-15
Circuito de un registro multifuncional.

Tabla 8-2
Registro multifunción.

S_1	S_0	Función
0	0	Carga paralelo
0	1	Puesta a 0 síncrona
1	0	Puesta a 1 síncrona
1	1	Desplazamiento Derecha

Otras funciones que pueden ser útiles en un registro son:

- Negar o voltear el contenido del registro.
- Desplazar entre las posiciones pares o impares del registro.
- Desplazar con recirculación.
- Puesta a 0 o a 1 síncronas.

El lector puede diseñar estos registros utilizando Mx 4:1 o similares.

8.8. Buses de datos

En general los registros no se encuentran aislados, sino que interactúan compartiendo un mismo canal de información, llamado BUS.

Por ejemplo, si en un sistema que dispone de cuatro registros que interactúan entre sí *todos con todos*, planteamos la comunicación uniendo la salida de cada registro a cada una de las restantes entradas, resulta una comunicación muy complicada. La solución pasa por utilizar una única línea de comunicación compartida por todos de forma ordenada. Dicha línea se denomina BUS.

En estos sistemas es muy útil un nuevo registro con línea de carga, que bien puede cargarse (Carga = 1) o no (Carga = 0). La tabla 8-3 muestra su comportamiento, mientras que en la figura 8-16 encontramos su esquema.

Figura 8-16
Circuito y bloque de un registro paralelo/paralelo con inhibición.

Tabla 8-3
Registro paralelo/paralelo con inhibición.

Carga	Función
0	Inhibición, no carga
1	Carga en paralelo

Basándonos en el anterior registro podemos plantear la comunicación entre cuatro registros RA, RB, RC y RD según la figura 8-17. Las señales F_1F_0 indican cuál es el registro fuente en la comunicación, mientras D_1D_0 indican cuál es el registro destino. Así pues, un registro aporta la información y otro la recibe, estableciéndose una comunicación entre ellos.

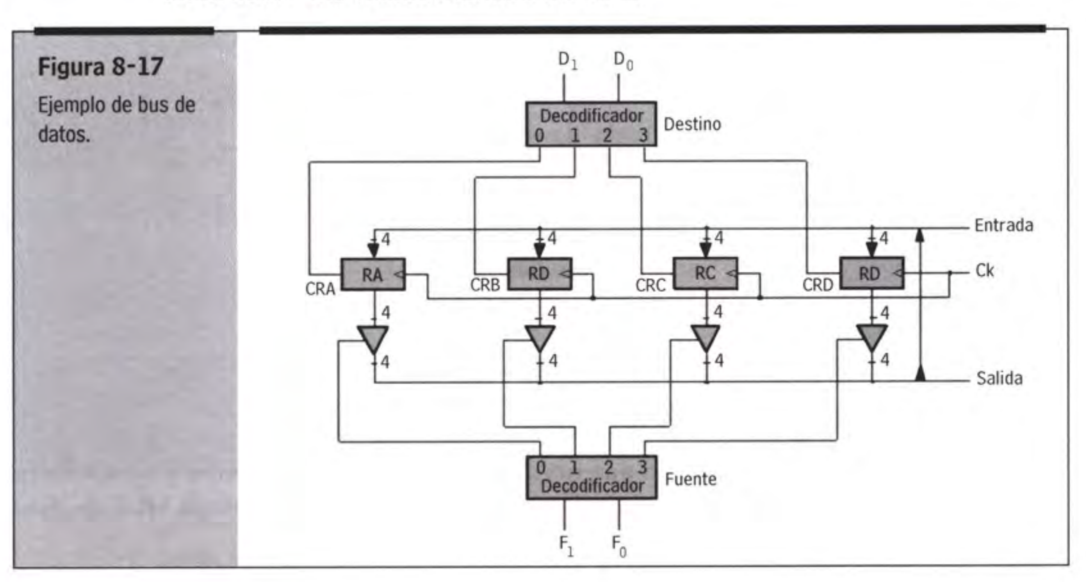


Figura 8-17
Ejemplo de bus de datos.

Por ejemplo, si $F_1F_0 = 10$ y $D_1D_0 = 00$, entonces de las salidas RA, RB, RC y RD sólo *pasará* el triestado la correspondiente a RC ($F_1F_0 = 10$), quedando el resto de las salidas aisladas de SALIDA, en alta impedancia (triestado). Así pues, al estar unidas las líneas de ENTRADA y SALIDA el contenido de RC se encuentra en la entrada de todos los registros, pero sólo se almacenará en RA, quedando el resto de los registros inhibidos, ya que $D_1D_0 = 00$.

Veamos otro ejemplo: grabar en cuatro registros una clave de cuatro dígitos BCD con sólo cuatro interruptores de entrada. El esquema resultante es el de la figura 8-17. Para cargar la clave 0870 habrá que poner 0000 en los cuatro interruptores conectados a ENTRADA y 00 en las líneas del decodificador de destino, de esta forma con el primer flanco RA se cargará con 0. Seleccionaremos ahora 1000 (8) en los interruptores y 01 en el decodificador, quedando cargado con un 8 el RB al llegar el flanco del reloj. El proceso se repite para los valores 0111 y 0000 y las líneas del decodificador 10 y 11. Pasados 4 flancos tendremos la clave al completo.

Como puede ver el lector, físicamente el BUS casi no existe; el BUS es un modo de conectar registros entre sí -o dispositivos de almacenamiento-, que tiene como base la inhibición y el triestado. La solución antes presentada no es única, existen otras, como la de la figura 8-18, que no necesitan de la línea de carga.

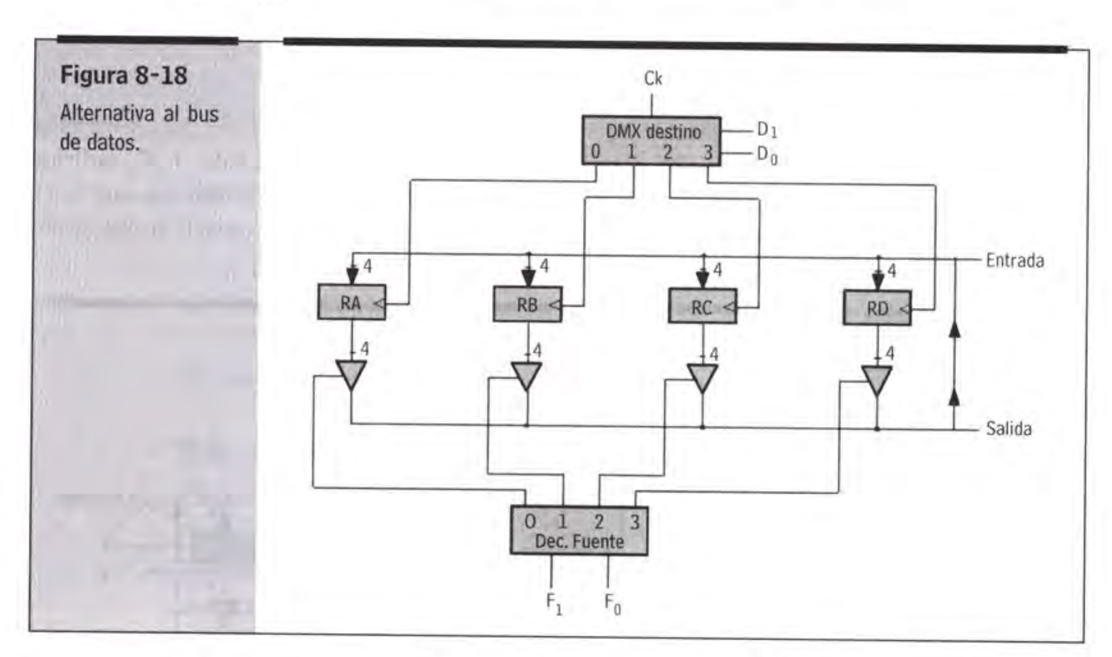


Figura 8-18
Alternativa al bus de datos.

8.9. Registros de desplazamiento tipo MOS

Los registros vistos con anterioridad son de tecnología bipolar y su diseño es principalmente lógico; sin embargo, existe también la tecnología MOS de diseño de dispositivos de almacenamiento.

 © ITES - PARANINFO

Los registros MOS tienen que ser con entrada y salida serie, y presentan como principal característica una alta densidad de integración y un consumo bajo.

El diseño de registros con tecnología MOS es básicamente hardware, manejando como elementos base el transistor MOS realimentado y las capacidades parásitas asociadas a dichos transistores. Las técnicas de diseño MOS exceden claramente los límites metodológicos de este libro, por lo que es suficiente aquí conocer su existencia.

8.10. Registros MSI

En este apartado describimos algunos de los registros disponibles en el mercado y su funcionamiento. En la tabla 8-4 se muestran los registros más comunes y útiles de la serie 74.

Tabla 8-4

Principales circuitos integrados con registros.

Dispositivo	Descripción	Sincronismo	Tamaño	Preset	Clear	Líneas auxiliares	Comentarios
74173	P-P con Línea de Carga (1)	↑	4 bits	No	Sí	ENO 1-0(b) ENI 1-0(b)	Triestado
74174	P-P	↑	6 bits	No	Sí		
74175	P-P	↑	4 bits	No	Sí		Q y $\overline{Q}$
74273	P-P	↑	8 bits	No	Sí		
74378	P-P con Línea de Carga	↑	6 bits	No	No	ENI(b)	
74373	Latch	Alto	8 bits	No	No	ENI(a) ENO(b)	Triestado
74374	P-P	↑	8 bits	No	No	ENO(b)	Triestado
7491	S-S	↑	8 bits	No	No	A B	Entrada=A·B
7496	S-S y P-S	↑	5 bits	Sí	Sí	PE(b)	Carga Asíncrona
74164	S-S y S-P	↑	8 bits	No	Sí	A B	Entrada=A·B
74165	P-S	↑	8 bits	No	No	ENI(b) $\overline{L}$/S	Carga y Desplaza
7495	P-P Dcha./Izqda.	↑ dos relojes	4 bits	No	No	DS	Modificación por Hw
74299	Dcha./Izqda.	↑	8 bits	No	Sí	ENI 1-0(b) S 1-0	Permite la carga Paralelo
74179	Universal	↑	4 bits	No	Sí	L(a) S(a)	Carga Síncrona
74194	Universal	↑	4 bits	No	Sí	S1-0	

(1) P-P significa Paralelo-Paralelo, S-S Serie-Serie, P-S Paralelo-Serie y S-P Serie-Paralelo.
(a) Significa que la señal es activa por nivel alto.
(b) Significa que la señal es activa por nivel bajo.

A continuación describiremos con más detalle algunos de los registros de la tabla 8-4.

8.10.1. Registro 7491

Registro de desplazamiento de 8 bits con entrada serie y salida serie (figura 8-19). Carece de línea de inicialización.

Es destacable que la entrada no es una sola línea ES, sino que se obtiene del producto de las líneas A y B.

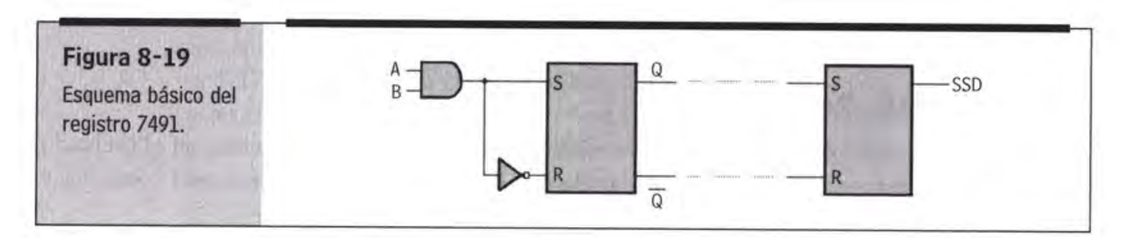


Figura 8-19
Esquema básico del registro 7491.

Una línea, por ejemplo B, puede verse como ENABLE, pues si B vale 0 el registro se carga con 0, y si B vale 1 el registro se cargará con el valor de A.

La tabla funcional y el bloque son los de la tabla 8-5 y la figura 8-20.

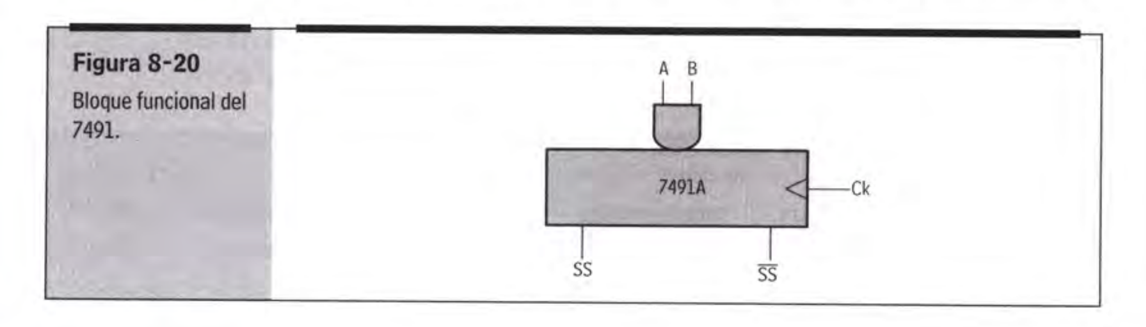


Figura 8-20
Bloque funcional del 7491.

Tabla 8-5
Tabla del registro 7491.

		tras 8 flancos	
A	**B**	**SS**	**$\overline{SS}$**
1	1	1	0
0	X	0	1
X	0	0	1

8.10.2. Registro 74164

Registro de desplazamiento de 8 bits con entrada serie y salida serie y paralelo (figura 8-21). Tiene una línea CLEAR común de inicialización; al igual que el anterior registro, la entrada serie se obtiene del producto de dos líneas A y B.

En la tabla de funcionamiento 8-6, cuando a Q_6 se le asigna Q_7 quiere decir que el valor actual de Q_6 es el anterior de Q_7: $Q_6(t) = Q_7(t-1)$, es decir, Q_7 se ha desplazado a Q_6. Lo dicho será válido en las siguientes tablas.

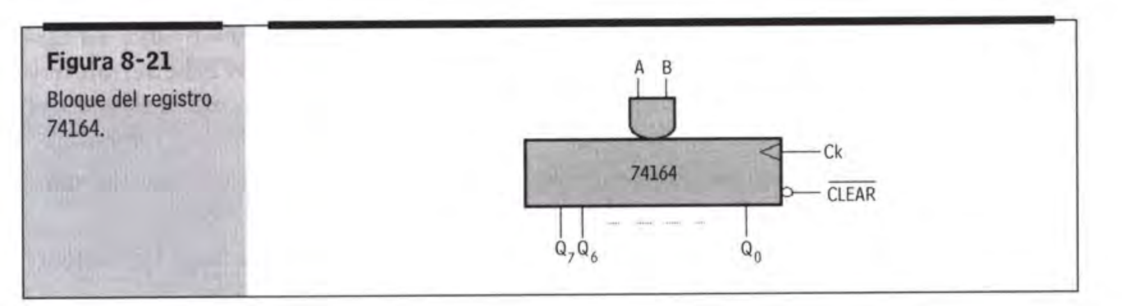


Figura 8-21
Bloque del registro 74164.

Tabla 8-6
Comportamiento del 74164.

$\overline{CLEAR}$	Ck	A	B	Q_7	Q_6	...	Q_0	Función
0	X	X	X	0	0	...	0	Inicializar
1	↑	1	1	1	Q_7	...	Q_1	Cargar
1	↑	0	X	0	Q_7	...	Q_1	Cargar
1	↑	X	0	0	Q_7	...	Q_1	Cargar
1	0	X	X	Q_7	Q_6	...	Q_0	Memoria

8.10.3. Registro 7496

Registro de desplazamiento de 5 bits con entrada serie y salida serie (figura 8-22). Tiene una línea CLEAR común asíncrona y dispone de cinco líneas de Preset asíncrono controladas por una línea Preset Enable (PE).

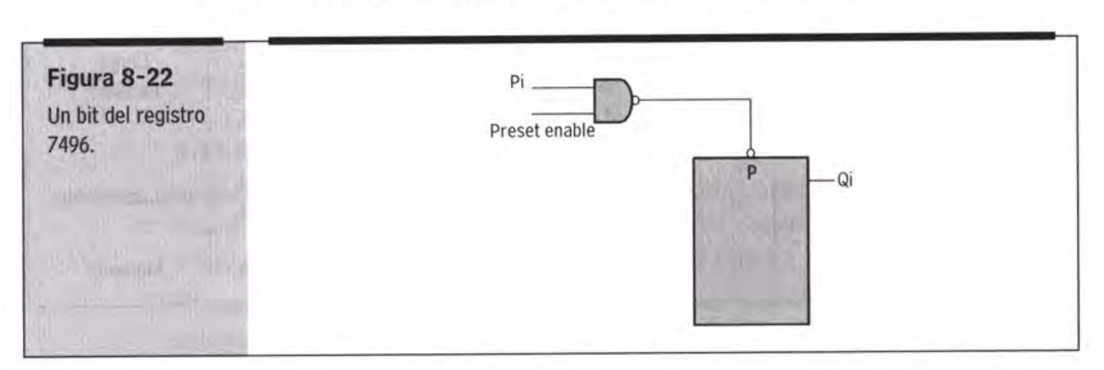


Figura 8-22
Un bit del registro 7496.

Las líneas de Preset de cada báscula se obtienen como la NAND entre la Preset correspondiente y la Preset Enable.

El funcionamiento es idéntico al 74164 excepto en el uso del Preset asíncrono, que permite una carga en paralelo siguiendo los siguientes pasos:

1. Activación de la línea $\overline{CLEAR}$ que pone todas las básculas a 0.
2. Poner en las líneas Pi los correspondientes valores a cargar en paralelo.

3. Generar un pulso de nivel alto en Preset Enable (0 - 1 - 0) y así cargar con 1 aquellas básculas cuyo Pi = 1. El pulso debe ser breve y anterior a la llegada del siguiente ↑ del reloj, para no entorpecer el desplazamiento.

Con esta secuencia primero se cargan los ceros a través de Clear y luego los unos a través de Pi y Preset Enable.

La tabla funcional y el bloque del 7496 se corresponden con la tabla 8-7 y la figura 8-23.

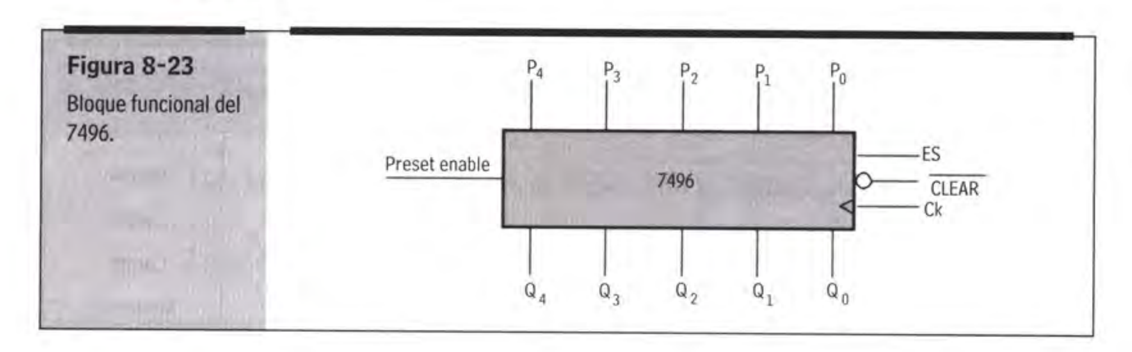


Figura 8-23
Bloque funcional del 7496.

Tabla 8-7
Comportamiento del 7496.

$\overline{CLEAR}$	Preset Enable	P_4	P_3	P_2	P_1	P_0	Ck	ES	Q_4	Q_3	Q_2	Q_1	Q_0	Función
0	0	X	X	X	X	X	X	X	0	0	0	0	0	Inicializ.
1	1	0	0	0	0	0	X	X	0	0	0	0	0	Carga Paralelo
1	1	1	1	1	1	1	X	X	1	1	1	1	1	
1	1	0	0	0	0	0	X	X	Q_4	Q_3	Q_2	Q_1	Q_0	
1	1	1	0	1	0	1	X	X	1	Q_3	1	Q_1	1	
1	0	X	X	X	X	X	↑	1	1	Q_4	Q_3	Q_2	Q_1	Desplazamiento
1	0	X	X	X	X	X	↑	0	0	Q_4	Q_3	Q_2	Q_1	
1	0	X	X	X	X	X	0	X	Q_4	Q_3	Q_2	Q_1	Q_0	Memoria

8.10.4. Registro 74165

Registro de desplazamiento de 8 bits con entrada serie o paralelo y salida serie. La entrada paralelo es asíncrona utilizando las líneas de Preset y Clear. La línea Desp./$\overline{\text{Carga}}$ controla la función del registro, si es 1 la entrada es serie y si es en 0 la entrada es en paralelo. Además, el 74165 dispone de una línea de Inhibición del Reloj, si está a 1 el Ck queda inhibido, debiendo estar a 0 para que el registro funcione con normalidad. El 74165 no dispone de línea de Clear independiente.

La tabla funcional y el bloque del 74165 se corresponden con la tabla 8-8 y la figura 8-24.

Tabla 8-8
Comportamiento del 74165.

Despl./ $\overline{CARGA}$	Inhib. Reloj	Ck	ES	Paralelo $D_7 .... D_0$	$Q_7 .... Q_0$	Q_0 SS	$\overline{Q_0}$ $\overline{SS}$	Función
0	X	X	X	$D_7 .... D_0$	$D_7 .... D_0$	D_0	$\overline{D_0}$	Carga Paralelo
1	1	X	X	X	$Q_7 .... Q_0$	Q_0	$\overline{Q_0}$	Registro Inhibido
1	0	-	1	X	$1 ..... Q_1$	Q_1	$\overline{Q_1}$	Desplazamiento
1	0	-	0	X	$0 ......Q_1$	Q_1	$\overline{Q_1}$	Desplazamiento
1	1	0	X	X	$Q_7 .... Q_0$	Q_0	$\overline{Q_0}$	Memoria

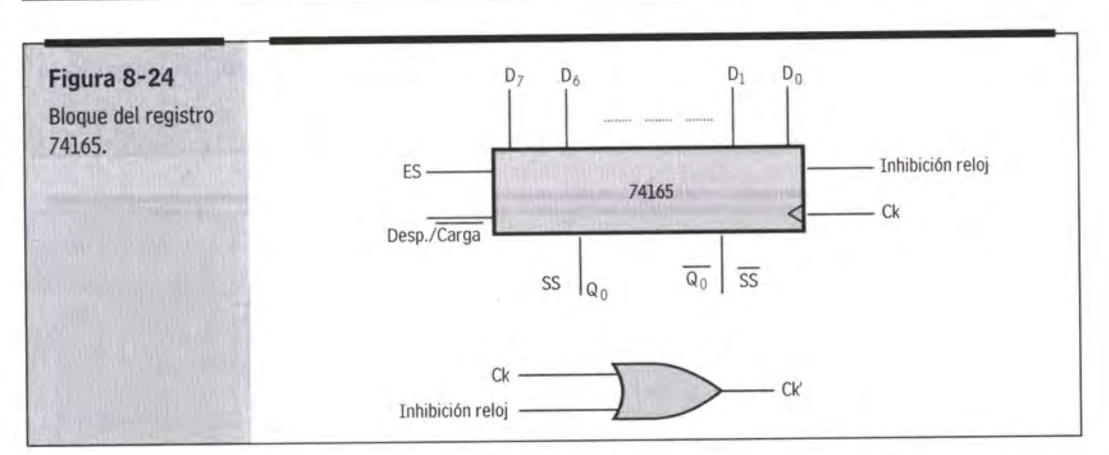


Figura 8-24
Bloque del registro 74165.

La estructura interna responde al circuito visto en el apartado de registros paralelo-serie con carga asíncrona utilizando las líneas de Preset y Clear.

8.10.5. Registro 74179

Registro de desplazamiento de 4 bits con entrada paralelo o serie y salida paralelo o serie. La entrada paralelo es síncrona. El registro dispone de una línea Clear común de inicialización.

El registro se comportará según el valor de las líneas Despl. y Carga, como muestra la tabla 8-9.

Tabla 8-9
Comportamiento del 74179.

Despl.	Carga	Función
0	0	Inhibición
0	1	Carga síncrona
1	0	Desplazamiento dcha.
1	1	Desplazamiento izqda.

Como podemos ver en la tabla 8-9, el 74179 además de tener carga paralelo o serie puede quedar inhibido síncronamente, es decir, aunque el registro reciba flancos su contenido no variará.

La tabla funcional y el bloque del 74179 se corresponden con la tabla 8-10 y la figura 8-25.

Tabla 8-10
Comportamiento del 74179.

$\overline{CLEAR}$	Ck	Desp	Carga	D_3	D_2	D_1	D_0	ES	Q_3	Q_2	Q_1	SS Q_0	Función
0	X	X	X	X	X	X	X	X	0	0	0	0	Borrado
1	↑	1	X	X	X	X	X	1	1	Q_3	Q_2	Q_1	Desplazamiento
1	↑	1	X	X	X	X	X	0	0	Q_3	Q_2	Q_1	Desplazamiento
1	↑	0	0	X	X	X	X	X	Q_3	Q_2	Q_1	Q_0	Inhibición
1	↑	0	1	D_3	D_2	D_1	D_0	X	D_3	D_2	D_1	D_0	Carga Paralelo
1	0	X	X	X	X	X	X	X	Q_3	Q_2	Q_1	Q_0	Memoria

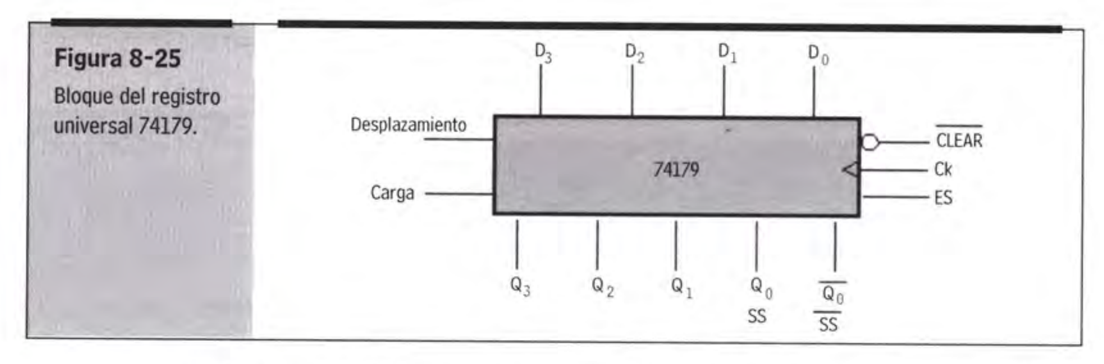


Figura 8-25
Bloque del registro universal 74179.

Si el registro 74165 respondía a la implementación con carga asíncrona, el 74179 responde a la implementación con carga síncrona, aquella que tenía como elemento de control el multiplexor.

8.10.6. Registro 74194

Registro universal de 4 bits con entrada paralelo o serie y salida paralelo o serie. Dispone de línea Clear común para inicialización asíncrona. El desplazamiento es bidireccional. El modo de funcionamiento depende de S_1 y S_0, como muestra la tabla 8-11.

Tabla 8-11
Comportamiento del 74194.

S_1	S_0	Función
0	0	Inhibición
0	1	Despl. Izquierda
1	0	Despl. Derecha
1	1	Carga en paralelo

La tabla funcional y el bloque del 74194 se corresponden con la tabla 8-12 y la figura 8-26.

Tabla 8-12
Comportamiento del 74194.

$\overline{CLEAR}$	Ck	S_1	S_0	D_3	D_2	D_1	D_0	ESD	ESI	SSI Q_3	Q_2	Q_1	SSD Q_0	Función
0	X	X	X	X	X	X	X	X	X	0	0	0	0	Inicialización
1	↑	0	0	X	X	X	X	X	X	Q_3	Q_2	Q_1	Q_0	Mov. Inhibición
1	↑	0	1	X	X	X	X	X	1	Q_2	Q_1	Q_0	1	Despl. Izda.
1	↑	0	1	X	X	X	X	X	0	Q_2	Q_1	Q_0	0	Despl. Izda.
1	↑	1	0	X	X	X	X	1	X	1	Q_3	Q_2	Q_1	Despl. Dcha.
1	↑	1	0	X	X	X	X	0	X	0	Q_3	Q_2	Q_1	Despl. Dcha.
1	↑	1	1	D_3	D_2	D_1	D_0	X	X	D_3	D_2	D_1	D_0	Carga Paralelo
1	0	X	X	X	X	X	X	X	X	Q_3	Q_2	Q_1	Q_0	Memoria

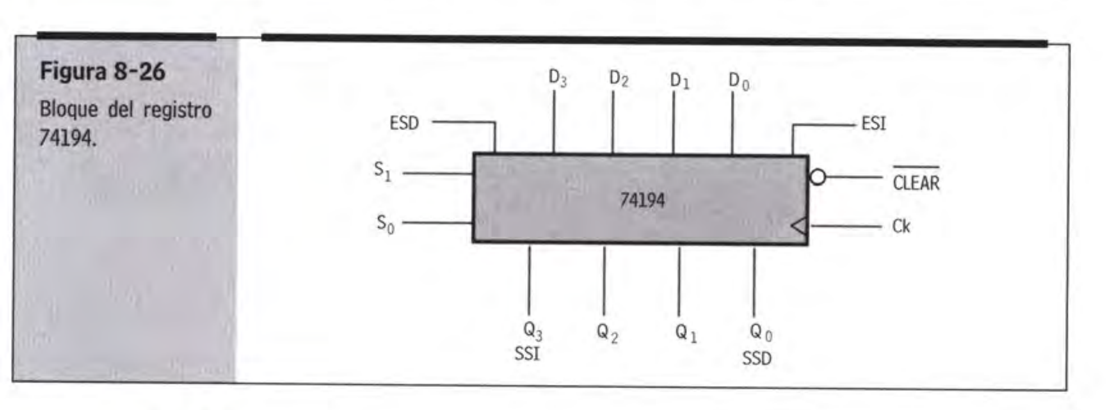
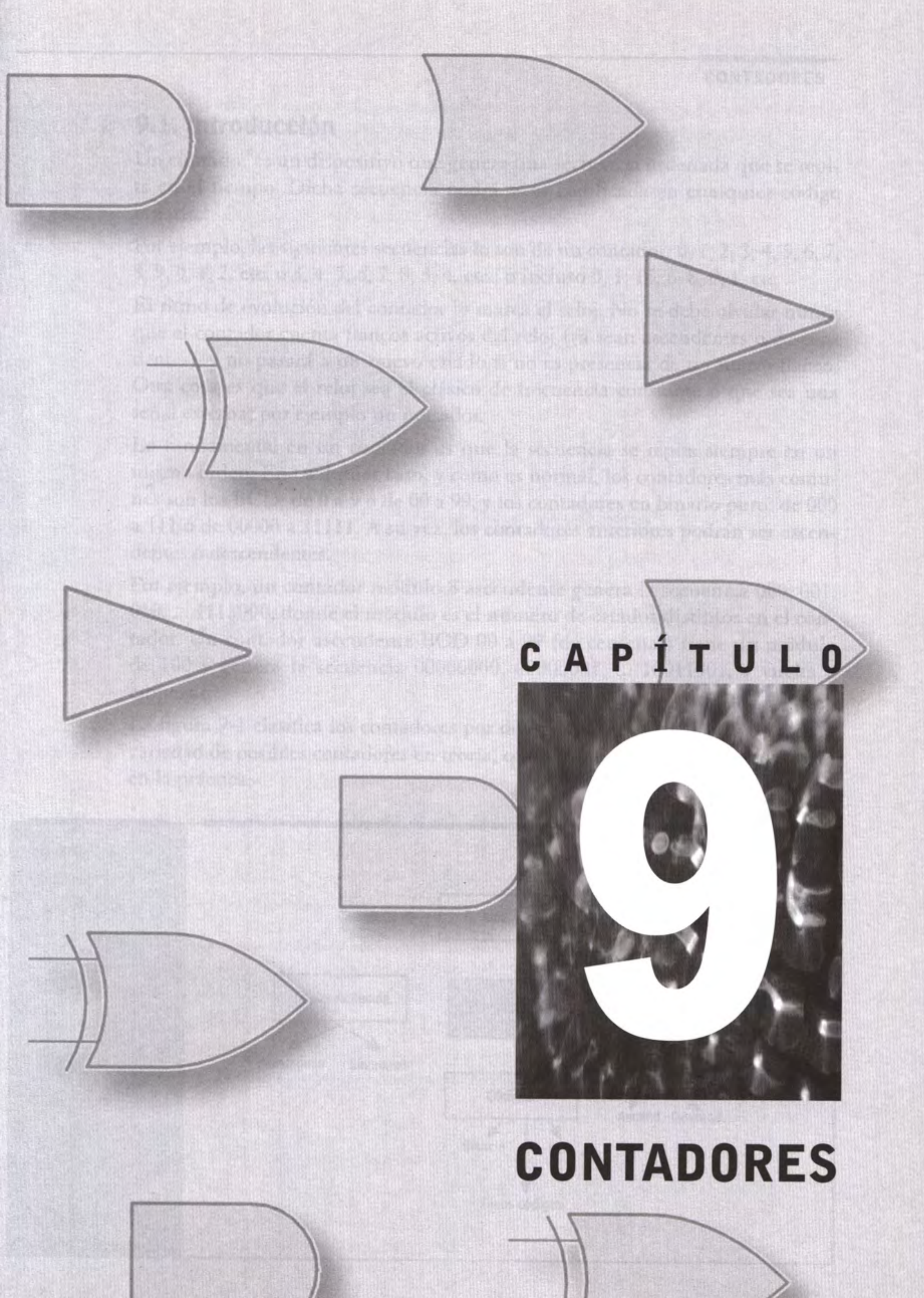


Figura 8-26
Bloque del registro 74194.

8.11. Resumen

En este capítulo hemos presentado el concepto de registro y su implementación. Tanto el uno como el otro son muy sencillos: el objetivo es guardar un dato, lo que se consigue gracias al biestable síncrono.

La diversidad de registros viene dada por cómo se recibe la entrada y/o se entrega la salida. De este modo el registro no sólo guardará un dato, sino que también adaptará distintos sistemas entre sí.



CAPÍTULO

9

CONTADORES

9.1. Introducción

Un contador es un dispositivo que genera una secuencia ordenada que se repite en el tiempo. Dicha secuencia podrá estar codificada en cualquier código binario.

Por ejemplo, las siguientes secuencias lo son de un contador: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, etc. o 3, 4, 5, 6, 7, 8, 3, 4, etc., o incluso 0, 1, 15, 6, 8, 0, 1, etc.

El ritmo de evolución del contador lo marca el reloj. No se debe olvidar nunca que el contador cuenta flancos activos del reloj (ya sean ascendentes o descendentes), y no pasará a un nuevo estado si no es presencia de un nuevo flanco. Otra cosa es que el reloj sea el clásico de frecuencia constante o que sea una señal externa; por ejemplo un pulsador.

Lo fundamental en un contador es que la secuencia se repita siempre en un mismo orden. En cualquier caso, y como es normal, los contadores más comunes son los BCD: de 0 a 9 o de 00 a 99, y los contadores en binario puro: de 000 a 111 o de 00000 a 11111. A su vez, los contadores anteriores podrán ser ascendentes o descendentes.

Por ejemplo, un contador módulo 8 ascendente genera la secuencia 000, 001, 010, ... 111, 000, donde el módulo es el número de estados distintos en el contador. Un contador ascendente BCD 00 a 99 (de centenas) tiene un módulo de 100 y genera la secuencia 00000000, 00000001, ... 10011001, y vuelta a empezar.

La figura 9-1 clasifica los contadores por diversos criterios. La figura destaca la variedad de posibles contadores en teoría, que al final se reducen drásticamente en la práctica.

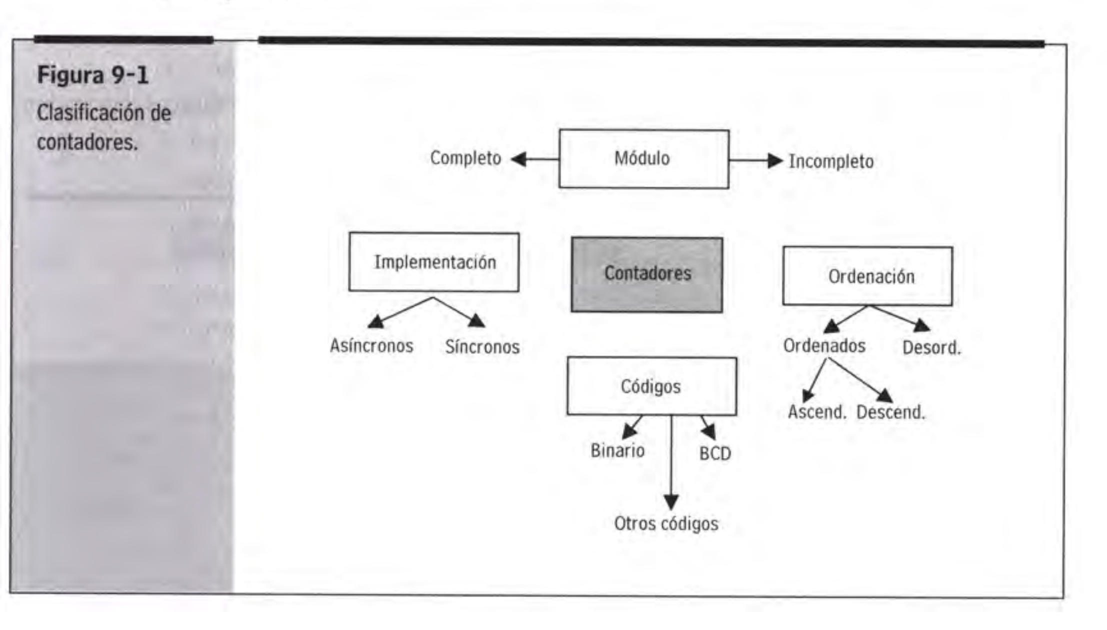


Figura 9-1
Clasificación de contadores.

En cuanto a los usos de un contador se puede decir lo mismo que para los registros, que son pocos y básicos, pero muy útiles. Un contador principalmente cuenta tiempo o eventos.

En los siguientes apartados describiremos los contadores, pero antes es necesario dividirlos en dos grupos: síncronos y asíncronos. Ambas técnicas de diseño obtienen los mismos contadores, pero desde planteamientos antitéticos, de manera que este capítulo, además de describir contadores, servirá para plantear y discutir cuestiones teóricas de los sistemas secuenciales.

Antes de empezar con los restantes apartados recuerde el lector que este capítulo, al igual que todos los demas, maneja dos puntos de vista. En el primer caso al lector sólo le interesa qué hace el contador, para qué usarlo y cómo usar un contador MSI. En el segundo caso el lector busca además de lo anterior conocer el cómo y el porqué del funcionamiento de un contador.

9.2. Contadores asíncronos

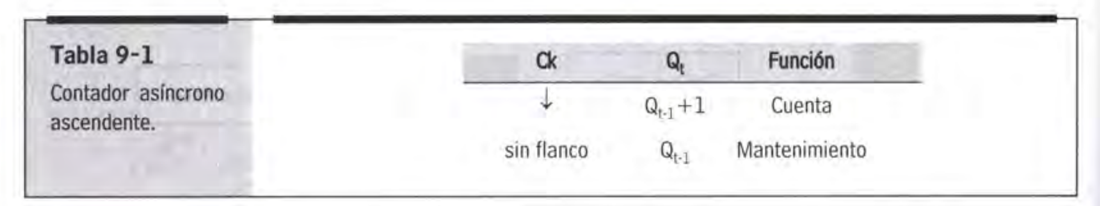
Un contador asíncrono (contador por propagación o ripple counter) genera la secuencia correspondiente con biestables J-K síncronos por flanco. Estos biestables evolucionan uno tras otro, de forma asíncrona; no todos a la vez. Los contadores asíncronos se distinguen porque cada flip-flop tiene una señal de reloj distinta, y sólo el primer flip-flop recibe la señal de reloj externa.

9.2.1 Definición

Un contador asíncrono ascendente de módulo completo genera la secuencia deseada al ritmo de los flancos activos del reloj.

9.2.2. Tabla

La tabla de verdad de un contador ascendente asíncrono por flancos descendentes es la tabla 9-1.


Tabla 9-1
Contador asíncrono ascendente.

Ck	Q_t	Función
↓	$Q_{t-1}+1$	Cuenta
sin flanco	Q_{t-1}	Mantenimiento

Podemos ver que la tabla es tan sencilla como la propia operación.

Una tabla más completa para un contador ascendente módulo 8 aparece en la tabla 9-2

Tabla 9-2
Contador módulo 8.

	t			t+1		
Ck	Q2	Q1	Q0	Q2	Q1	Q0
↓	0	0	0	0	0	1
↓	0	0	1	0	1	0
↓	0	1	0	0	1	1
↓	0	1	1	1	0	0
↓	1	0	0	1	0	1
↓	1	0	1	1	1	0
↓	1	1	0	1	1	1
↓	1	1	1	0	0	0
↓	0	0	0	0	0	1
↓	0	0	1	0	1	0
↓	..	..	..	..	..	..

9.2.3. Circuito lógico

Para diseñar un contador asíncrono ascendente de módulo completo (potencia de 2: 2, 4, 8, 16, etc.) síganse estos pasos:

1. Elegir el número necesario de flip-flop's J-K, que en principio tendrán que ser síncronos por flanco descendente.
2. Pónganse todas las entradas J y K a '1': $J_i = K_i = 1$.
3. Conéctese la señal externa de reloj al primer biestable: $Ck_0 = Ck$.
4. Conéctese la entrada de reloj de los restantes biestables a la salida del anterior biestable $Ck_i = Q_{i-1}$.
5. Las salidas del contador se obtienen ordenadas de izquierda a derecha de las salidas de los propios biestables.
6. Añádase la línea de INICIO utilizando las líneas de CLEAR y PRESET de cada biestable (en este caso CLEAR).

Aplicando los anteriores pasos, la figura 9-2 muestra el circuito lógico de un contador asíncrono ascendente módulo 16 activo por flancos descendentes.

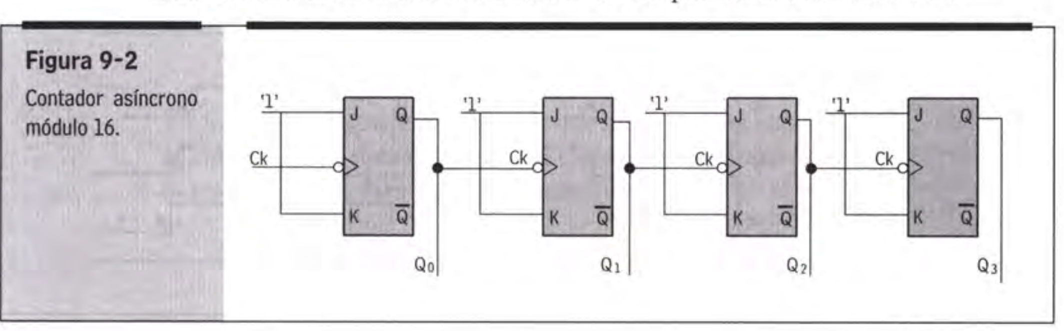


Figura 9-2
Contador asíncrono módulo 16.

9.2.4. Cronograma

Analicemos el circuito anterior obteniendo su cronograma. Para completar la figura 9-3 observemos que los biestables sólo bascularán (J=K=1) si la salida del anterior biestable ha pasado de 1 a 0, generando un flanco descendente.

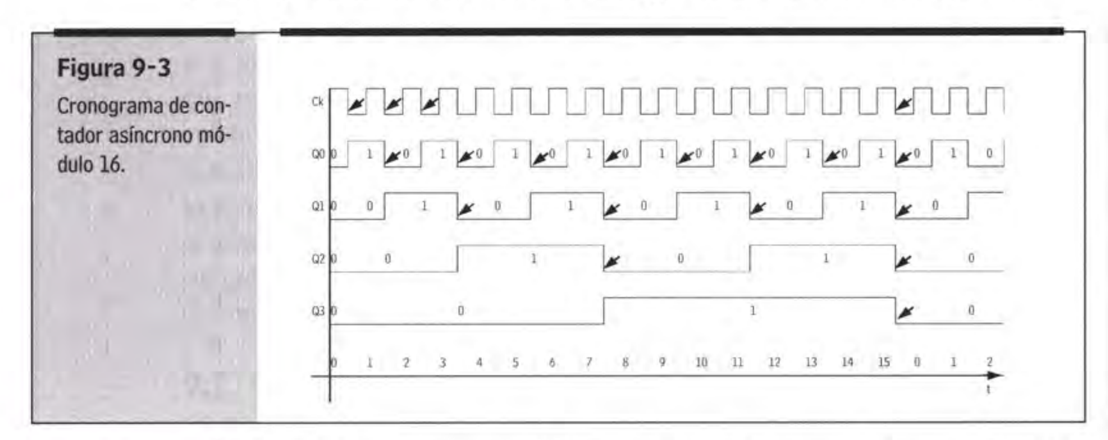


Figura 9-3
Cronograma de contador asíncrono módulo 16.

A la vista del cronograma es claro que el contador genera la secuencia 0000, 0001, 0010, 1111, ..., 0000, etc. Los distintos flancos marcados destacan que cada biestable es arrastrado o gobernado por el anterior, resultando en conjunto una estructura asíncrona: los biestables no evolucionan a la vez (síncronamente), sino de uno en uno (asíncronamente).

Queda también claro que las diferentes salidas Q0, Q1, Q2 y Q3 son también relojes. Q0 tiene una frecuencia que es la mitad de la del reloj externo Ck; siendo la de Q3 la dieciseisava parte de la de Ck. Así, por ejemplo, si Ck tuviese una frecuencia de 1 KHz, Q3 tendría una frecuencia de 66 Hz. O si Ck tuviera una frecuencia de 50 Hz (la de la red) y el contador fuera de módulo 50, la línea de más peso de la salida tendría una frecuencia de 1 Hz. Es decir, un contador es un divisor de frecuencias.

Planteemos ahora en la figura 9-4 un cronograma un poco más elaborado. En él incluiremos los retardos de cada biestable y la línea INICIO. El contador planteado es de módulo 8, veamos en la tabla 9-3 su funcionamiento.

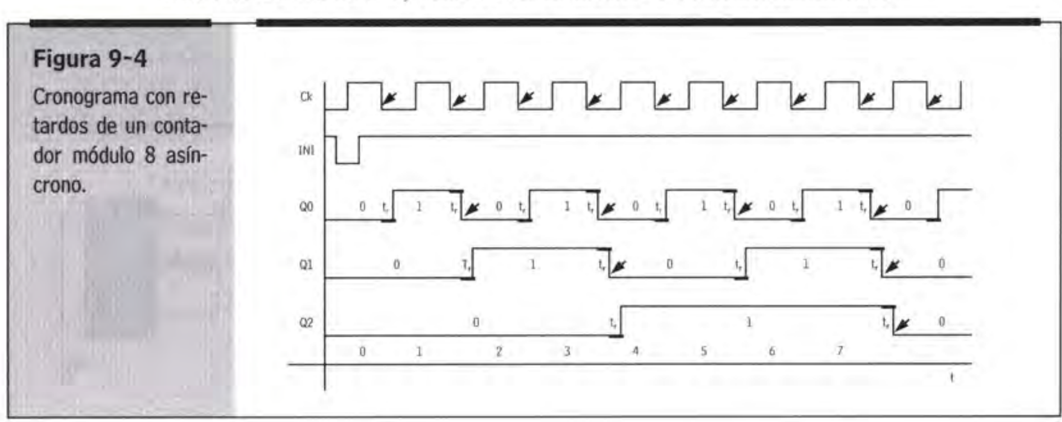


Figura 9-4
Cronograma con retardos de un contador módulo 8 asíncrono.

Tabla 9-3
Contador módulo 8 con INICIO.

t					t+1			
Ck	INI	Q2	Q1	Q0	Q2	Q1	Q0	Operación
	0	0	0	0				Inicio
↓	1	0	0	0	0	0	1	Primer flanco
↓	1	0	0	1	0	1	0	Segundo flanco
↓	1	0	1	0	0	1	1	Tercer flanco
↓	1	0	1	1	1	0	0	Cuarto flanco
↓	1	1	0	0	1	0	1	Quinto flanco
↓	1	1	0	1	1	1	0	Sexto flanco
↓	1	1	1	0	1	1	1	Séptimo flanco

Véase en la tabla 9-3 que el contador cuenta los flancos ya recibidos, y no los flancos menos 1 como se indica en algunos textos. En un contador es básica la inicialización.

En la figura 9-4 hemos marcado exageradamente el retardo asociado a cada biestable. Así, desde que el Ck *despierta* con su flanco descendente a Q0 hasta que éste se pone a 1 pasa un tiempo tpd (tr en la gráfica). Lo destacable es que por ser una estructura asíncrona estos retardos se acumulan (Ck despierta a Q0, éste a Q1 y éste a Q2), de manera que el retardo del contador es 3 x tpd en el peor de los casos:

$$t\ evolucion = 3 \times tpd$$

La expresión anterior no es completa, pues el tiempo tpd habría que sumarlo con tsetup y thold (tiempo de establecimiento y de mantenimiento de las entradas J y K); ahora bien, como J=K=1 de continuo ambos tiempos no deben ser acumulados a tpd. Generalizando para un contador de módulo 2^n (con n biestables) el retardo es:

$$tevol = n \times (tpd + tsetup + thold) = n \times tpd$$

Si n es un número bajo el valor del retardo total resulta despreciable, pero no así si n es número alto. La frecuencia del reloj queda por tanto limitada.

$$fmax\ Ck = \frac{1}{n \times tpd}$$

Es decir, Ck no puede entregar un nuevo flanco si el contador *no ha acabado* con el anterior. Esta restricción práctica, y otras, hacen que se usen poco los contadores asíncronos para valores elevados de n.

9.2.5. Glitches

Volviendo a la figura 9-4, podemos ver que el contador no pasa del 111 al 000 instantáneamente. La secuencia es la siguiente:

ESTADO ORIGEN	111
	110
	100
ESTADO FINAL	000

O sea, durante un tiempo el contador presenta 3 estados espurios que no son reales (pasa por 7, 6, 4 y 0). Estos glitches pueden llevar a otras partes del diseño a un mal funcionamiento. Este comportamiento, junto con la restricción impuesta a la frecuencia, hace que los contadores asíncronos sean poco usados; cada vez menos.

9.2.6. Bloque

El aspecto general de un contador asíncrono se muestra en la figura 9-5.

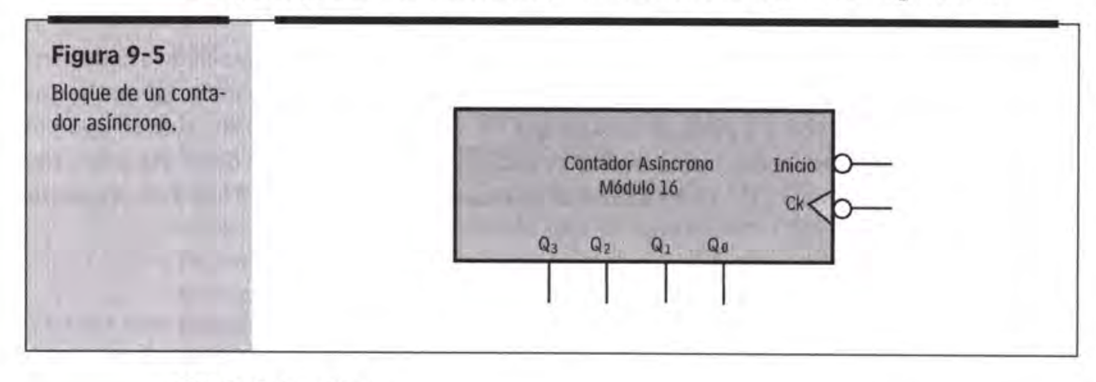


Figura 9-5
Bloque de un contador asíncrono.

9.2.7. Extensión

Si en vez de un contador módulo 16, lo quisiéramos de módulo 1.024, sólo habría que repetir lo hecho en el esquema de la figura 9.2, pero ahora con 10 flip-flop's.

La extensión del contador asíncrono no es fácil, es directa.

9.2.8. Líneas auxiliares

De momento hablemos de INICIO (más adelante veremos otras líneas auxiliares). Lo más normal es que sea activo por nivel bajo y que esté conectado al CLEAR de los biestables; al activarse fuerza el valor 0000 en el contador. También podría forzar el valor 1111 si estuviera conectado al PRESET, incluso un contador puede tener dos líneas de inicio: una de puesta a 0 y otra de puesta a 1. En cualquier caso, la línea INICIO es básica en un contador, ya que nos asegura que empezará a contar en un estado conocido y deseado, o que perdida la secuencia la recuperaremos a un valor conocido.

9.2.9. Contador descendente

Dentro de la estructura asíncrona, diseñar un contador descendente tiene varias opciones.

En primer lugar, en vez de leer la salida de Qi bastaría con leerla en Qi negada. De esta forma, la secuencia generada pasaría de 000, 001,..., 111 a 111, 110, 101,..., 000.

Otra forma de diseñar un contador descendente consiste en simplemente cambiar los biestables J-K por otros que sean síncronos por flanco ascendente. El esquema de un contador descendente módulo 8 es el de la figura 9-6.

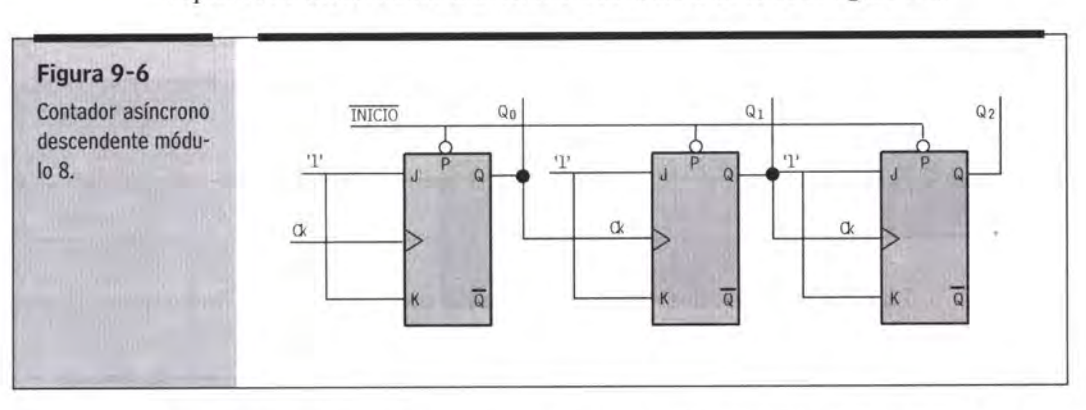


Figura 9-6
Contador asíncrono descendente módulo 8.

Queda para el lector obtener el cronograma. Es interesante completarlo, ya que se ve que con sólo cambiar la sincronía cambiará la secuencia, ahondando en el carácter asíncrono del diseño presentado.

La anterior solución es válida, pero en principio no tiene sentido contar flancos descendentes en un caso y ascendentes en el otro. La solución más utilizada pasa por conectar la salida negada al Ck de cada biestable, como en el circuito lógico de la figura 9-7.

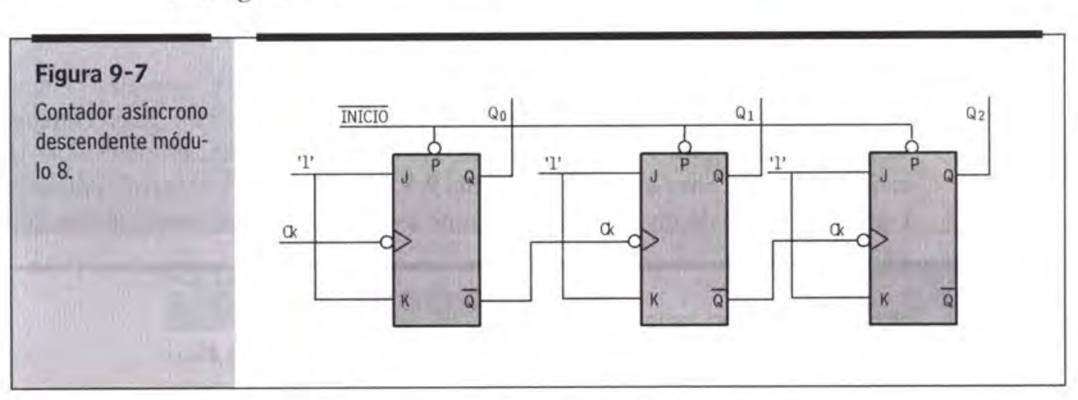


Figura 9-7
Contador asíncrono descendente módulo 8.

Para obtener el cronograma de la figura 9-8 hay que tener en cuenta que la salida está en Qi, pero quien gobierna el contador es la Qi negada.

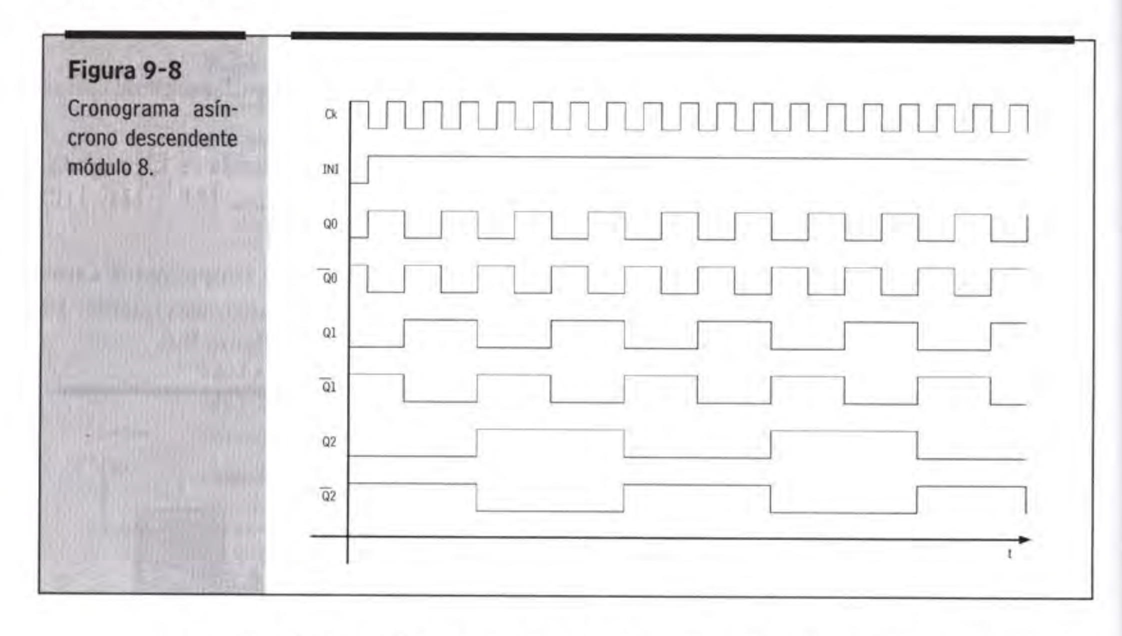


Figura 9-8
Cronograma asíncrono descendente módulo 8.

La tabla 9-4 intenta ordenar lo dicho. En cada celda queda dicho qué contador resulta según qué línea conectemos a Ck y qué flanco será el activo.

Tabla 9-4
Ordenación de contadores asíncronos.

Tipo de J-K	Unir Q con Ck	Unir $\overline{Q}$ con Ck
Activo por ↑	Descendente	Ascendente
Activo por ↓	Ascendente	Descendente

9.2.10. Contador Ascendente/Descendente

Este contador es capaz de generar una secuencia ascendente o descendente según el valor de una línea MODO, así, por ejemplo, si MODO=0 cuenta ascendente, y viceversa para MODO=1.

En el punto anterior hemos visto que un contador ascendente y otro descendente se distinguen en la línea que conectan al reloj de cada biestable. En el circuito lógico de la figura 9-9 el multiplexor 2:1 se encarga de encaminar al Ck de cada biestable la línea adecuada. La tabla 9-5 y las figuras 9-9 y 9-10 describen el comportamiento de un contador asíncrono ascendente/descendente módulo 16.

Tabla 9-5
Contador ascendente/descendente.

Ck	MODO	Qt	Función
↓	0	$Q_{t-1}+1$	Cuenta ASC
↓	1	$Q_{t-1}-1$	Cuenta DESC
sin flanco	X	Q_{t-1}	Mantenimiento

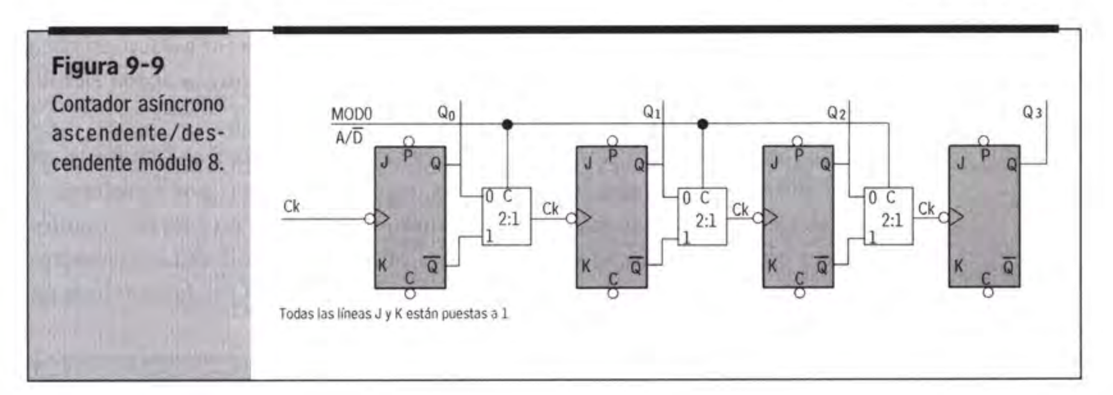


Figura 9-9
Contador asíncrono ascendente/descendente módulo 8.

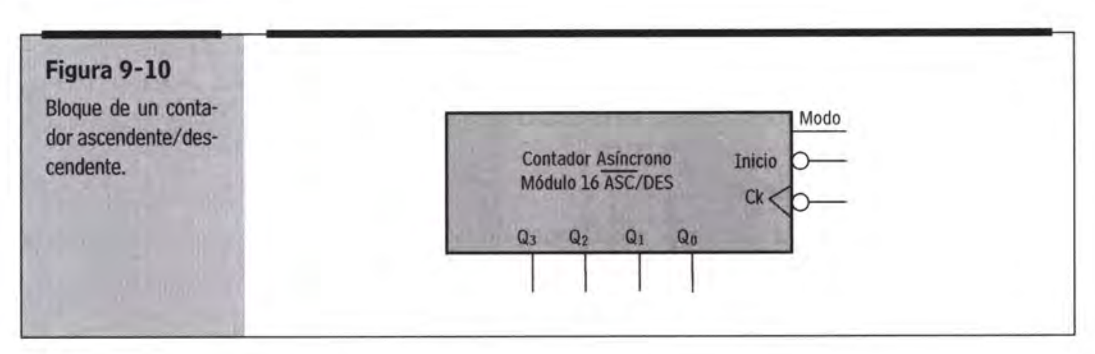


Figura 9-10
Bloque de un contador ascendente/descendente.

En cuanto al INICIO, puede optarse por dos opciones:

- Incluir dos señales de inicio independientes entre sí: una de inicio con 0's y otra de inicio con 1's. El usuario debe saber cuál activar en cada momento.
- Incluir una única señal de INICIO que active el CLEAR o el PRESET de los biestables según sea el valor de la línea MODO (figura 9-11).

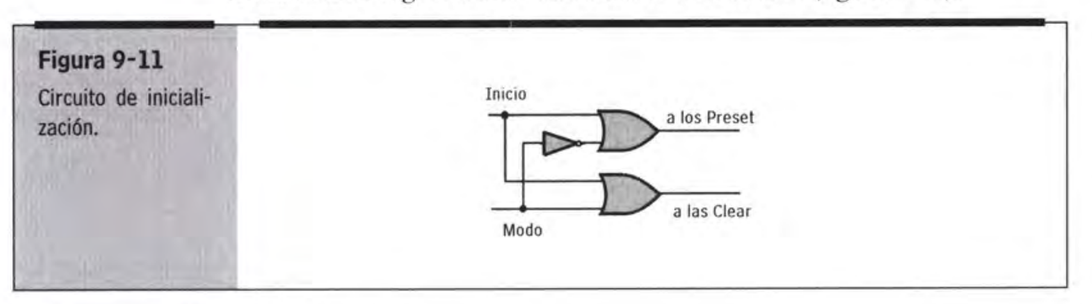


Figura 9-11
Circuito de inicialización.

9.2.11. Comentarios

Un contador asíncrono es muy fácil de diseñar, pero tiene inconvenientes teóricos y prácticos.

En la práctica un contador asíncrono sólo puede ser usado con facilidad para los códigos binario puro y BCD, resultando muy incómodo para otros códigos. Ahora bien, esta restricción no es importante, toda vez que la mayoría de las secuencias útiles están codificadas en binario o BCD.

Otra desventaja práctica viene de la imposición que un contador asíncrono hace a la frecuencia del Ck, pero en muchos casos esta imposición no acaba siendo una restricción real.

Por último, hay una restricción teórica. Cada vez más los diseños tienden a ser síncronos puros, sin más señales asíncronas que las impuestas por el sistema, e incluso éstas suelen acabar más o menos sincronizadas. De esta forma, mantener el uso y estudio de contadores asíncronos puede ser considerado un anacronismo que no debe ser inculcado a los nuevos profesionales. Queda esta duda en manos del lector y del profesional.

9.3. Contadores síncronos

Los contadores síncronos generan la secuencia del contador con biestables síncronos por flanco. Estos biestables evolucionarán todos a la vez, síncronamente, al contrario de los contadores asíncronos.

9.3.1. Definición

Un contador síncrono ascendente de módulo completo genera la secuencia deseada al ritmo de los flancos activos del reloj.

9.3.2. Tabla

La tabla 9-6 describe el funcionamiento de cualquier contador síncrono ascendente.

Tabla 9-6
Contador síncrono ascendente.

Ck	Qt	Función
$\downarrow$ (o $\uparrow$)	$Q_{t-1}+1$	Cuenta
sin flanco	Q_{t-1}	Mantenimiento

La tabla 9-1 del contador asíncrono y la tabla 9-6 del contador síncrono son idénticas, abundando en que ambos contadores generan la misma secuencia (mismas tablas), pero de distinto modo (distintos circuitos lógicos).

9.3.3. Circuito lógico

Para diseñar un contador síncrono de módulo completo hay que seguir los siguientes pasos:

1. Elegir el número adecuado de biestables J-K, cuyo sincronismo será elegido libremente por flanco ascendente o descendente.
2. Conéctese la señal externa de reloj a las entradas de reloj de todos los biestables: $Ck_i = Ck$
3. Conéctense las entradas J y K del primer flip-flop a 1: $J0 = K0 = 1$
4. Conéctense las entradas de los siguientes flip-flop's al producto de las salidas anteriores:

$$J_i = K_i = Q0 \cdot Q1 \cdot Q2 \cdot ... \cdot Q_{i-1}$$

5. Las salidas del contador se obtienen ordenadas de izquierda a derecha de las salidas de los propios biestables.
6. Añádase la línea de INICIO utilizando las líneas de CLEAR y PRESET de cada biestable (en este caso CLEAR).

Si seguimos los pasos anteriores para diseñar un contador síncrono ascendente módulo 8 activo por flancos ascendentes, obtendremos el circuito de la figura 9-12.

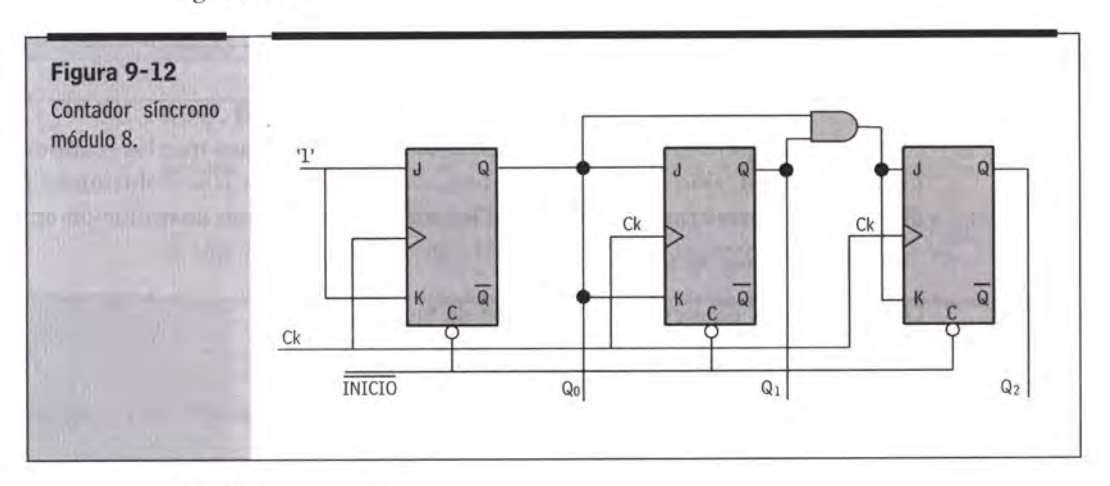


Figura 9-12
Contador síncrono módulo 8.

9.3.4 Cronograma

Obtengamos la figura 9-13 con el cronograma del circuito de la figura 9-12, teniendo en cuenta que todos los biestables evolucionan a la vez según el valor presente en sus entradas J y K antes del flanco activo.

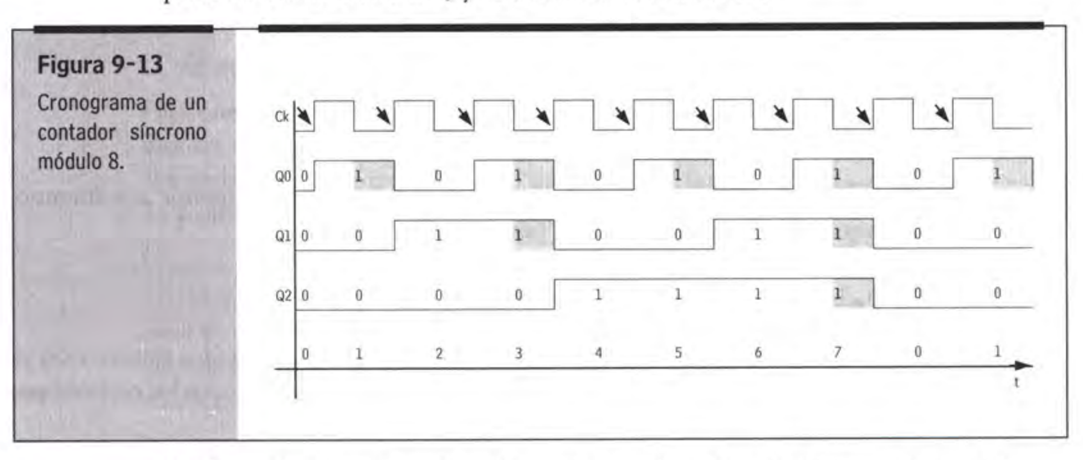


Figura 9-13
Cronograma de un contador síncrono módulo 8.

La figura 9-13 confirma la validez del circuito obtenido. Veamos ahora la figura 9-14 con el cronograma que incluye los retardos de los biestables.

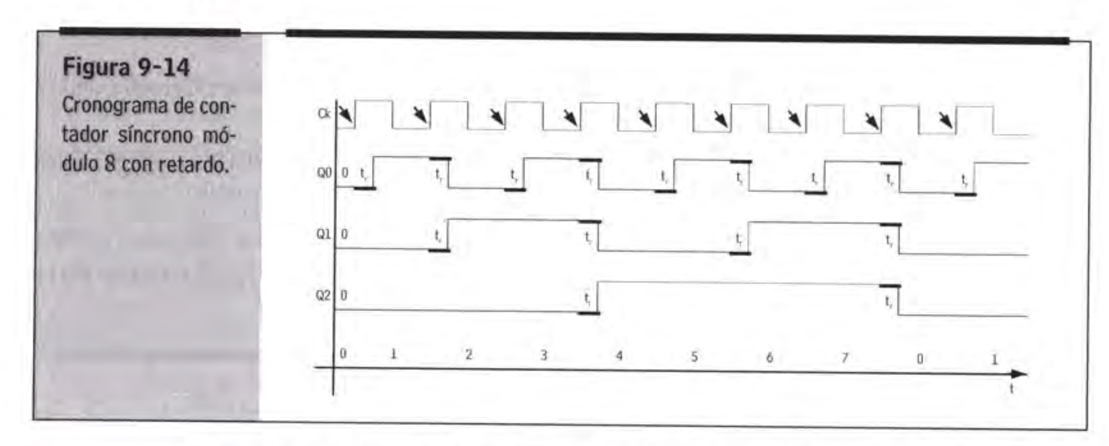


Figura 9-14
Cronograma de contador síncrono módulo 8 con retardo.

Si nos fijamos particularmente en el paso de 011 a 100 vemos que los retardos no se acumulan, sino que se superponen, ya que todos los J-K evolucionan a la vez y no se arrastran unos a otros. De este modo el tiempo de evolución es:

$$\text{tevol} = \text{tpd} + \text{tand}$$

Generalizando, para cualquier número de biestables:

$$\text{tevol} = \text{tpd} + \text{tand}$$

Vemos que ambas expresiones son idénticas, ya que no dependen de n, contrariamente a lo que ocurría con los asíncronos.

En los contadores síncronos sí hay que tener en cuenta los tiempos tsetup y thold. El primero no es problema ya que tpd > thold, luego los valores de J y K estarán presentes el tiempo suficiente tras el flanco (su salida no cambiará hasta pasado un tiempo tpd):

$$\text{tevol} = \text{tpd} + \text{tand} + \text{tsetup}$$

Reordenando lo anterior desde el punto de vista de la frecuencia:

$$\text{fmax Ck} = \frac{1}{\text{tpd} + \text{tand} + \text{tsetup}}$$

Resulta claro que en este aspecto el contador síncrono es superior al asíncrono, aunque todavía quedan puntos por discutir.

9.3.5. Glitches

En este caso el contador pasa de 0111 a 1000 sin producir estados transitorios, ya que todos los biestables basculan a la vez. De este modo el contador no tiene por qué influir negativamente en el resto del sistema.

9.3.6. Bloque

El aspecto externo de un contador síncrono es idéntico al de un contador asíncrono; su función es la misma (figura 9-15).

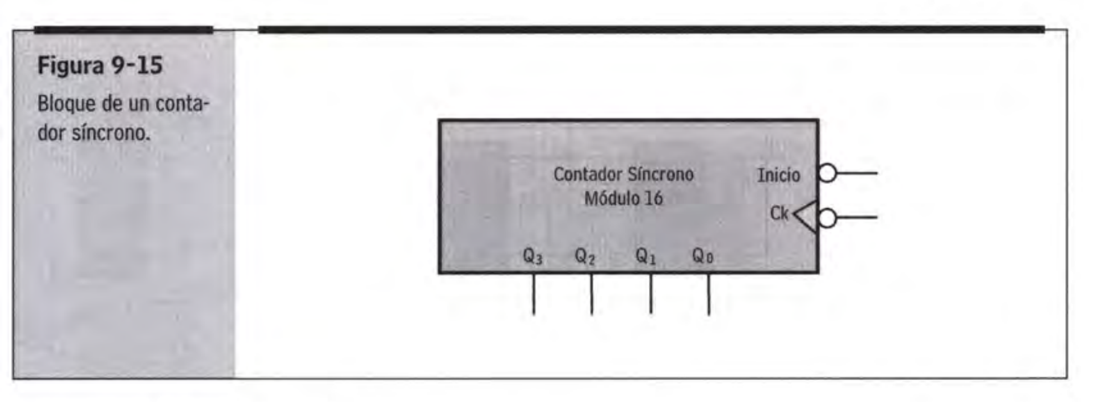


Figura 9-15
Bloque de un contador síncrono.

9.3.7. Extensión

En este punto aparecen de nuevo las diferencias entre síncrono y asíncrono.

Para diseñar un contador módulo 16 nos bastaría con añadir un J-K y una AND de tres entradas. El circuito lógico resultante aparece en la figura 9-16.

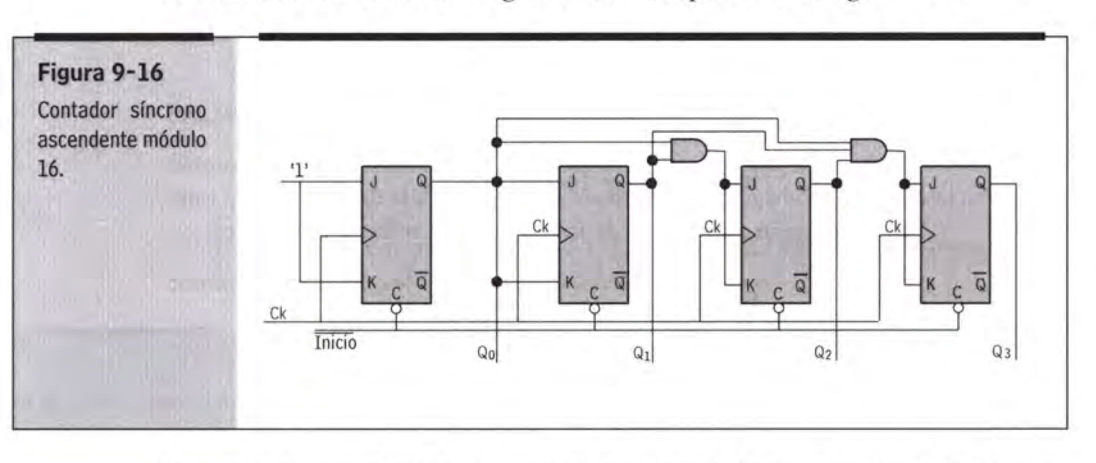


Figura 9-16
Contador síncrono ascendente módulo 16.

Para este circuito el tiempo de evolución ya calculado no variará. De hecho, para diseñar un contador de módulo 1.024 bastará con añadir 6 puertas AND y J-K. Sin embargo, este circuito tiene el problema del fan-in y del fan-out en la entrada y salida de la puerta AND, respectivamente.

Veamos, para un módulo 1.024 la última puerta AND tendría 8 entradas (fan-in), lo que no es ni común ni recomendable. Y además, la salida Q0 debería *atacar* a 9 entradas, lo que podría superar su fan-out. Es decir, la extensión es posible sobre el papel, pero no en la práctica por cuestiones tecnológicas (pensemos en un número n todavía mayor).

Para resolver el problema del fan-in y fan-out se plantea el contador síncrono con acarreo, cuyo circuito lógico aparece en la figura 9-17.

Figura 9-17
Contador síncrono ascendente con acarreo módulo 16.

Desde un punto de vista lógico los circuitos 9-16 y 9-17 son idénticos, pero su implementación es distinta. Ahora todas las AND tendrán dos entradas y el fan-out de cada salida no será más de dos.

Todo parece perfecto, pero hay un inconveniente: el tiempo de evolución. Ahora las puertas AND están en serie y por tanto para el contador módulo 16 resulta:

$$tevol = tpd + 2 \times tand$$

que generalizando se convierte en:

$$tevol = tpd + (n\text{-}2) \times tand$$

La restricción respecto de la frecuencia del reloj de entrada es:

$$fmax\ Ck = \frac{1}{tpd + (n\text{-}2) \times tand}$$

Es decir, aparece la n, cuya ausencia era el principal logro del contador síncrono. Eso sí, el valor n no afecta a tpd, sino a tand, con lo que su efecto cuantitativo es menor, se diluye. Resumiendo, cada estrategia tiene pros y contras.

9.3.8. Líneas auxiliares

Resaltar y recordar lo ya dicho para contadores asíncronos: todo contador debe tener una línea de INICIO.

9.3.9. Contador descendente

Obviando la solución ya dada de leer la salida en la Q negada, veamos cómo diseñar un contador descendente síncrono.

La solución pasa por conectar las entradas J y K de cada biestable al producto de las anteriores salidas negadas. La figura 9-18 muestra el circuito lógico de un contador síncrono descendente módulo 8.

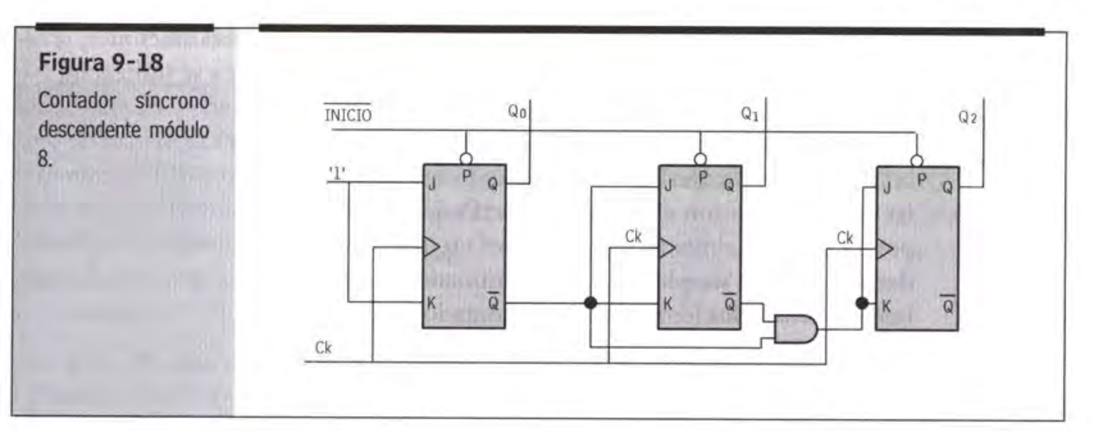


Figura 9-18

Contador síncrono descendente módulo 8.

Recuérdese que para diseñar un contador síncrono activo por flanco descendente bastará con sustituir los J-K por otros con esa sincronía, sin modificar nada más. No hay que confundir el sentido de la cuenta (ascendente o descendente) con los flancos que cuentan (ascendentes o descendentes), aunque se usen los mismos adjetivos.

9.3.10. Contador Ascendente/Descendente

Este contador dispone de una línea MODO que indicará el sentido de la cuenta, por ejemplo con MODO=0 cuenta ascendente y con MODO=1 descendente.

El diseño se basa en multiplexores 2:1. La figura 9-19 es el circuito lógico de un contador síncrono ascendente/descendente módulo 8.

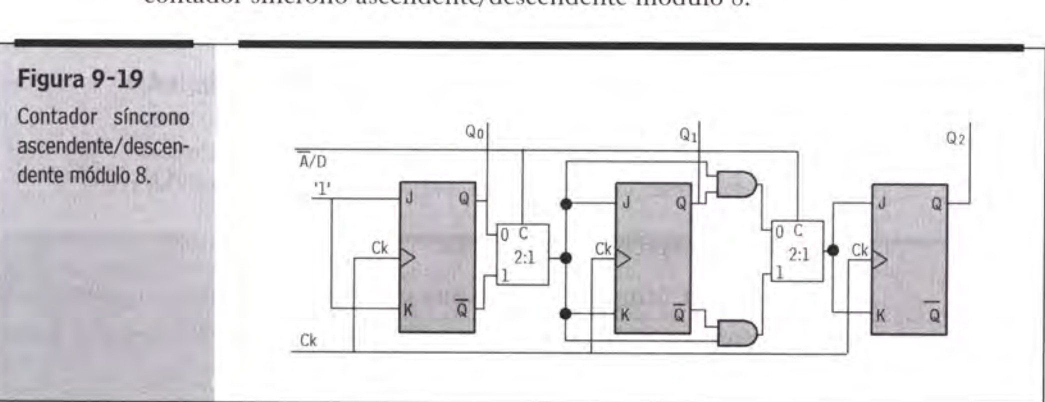


Figura 9-19

Contador síncrono ascendente/descendente módulo 8.

En cuanto a la línea o líneas de INICIO vale lo dicho para los contadores asíncronos ascendente/descendente.

9.3.11. Comentarios

Ya se han hecho ciertas consideraciones para los contadores asíncronos, sólo queda recalcarlas.

Muchos profesores (cada vez más) explican sólo los contadores síncronos, ocultando totalmente los asíncronos, así se aseguran de que nunca se utilizarán. No sólo eso, sino que se aseguran de que sus alumnos no tengan *ideas asíncronas*. Esta situación es idéntica a la planteada en los años ochenta en la enseñanza de programación informática. En aquel caso el comando GOTO (o JMP) se enfrentaba directamente con el paradigma de Programación Estructurada (que se tiene por ideal). Actualmente, los profesores no indican a sus alumnos la poca idoneidad del GOTO, simplemente han eliminado el comando de los temarios. Es más, buena parte de los lenguajes han eliminado esta instrucción de su repertorio.

Históricamente se empezó fabricando contadores asíncronos gracias a su sencillez, luego las mejoras del proceso de integración permitieron fabricar contadores síncronos; la pregunta es: ¿desaparecerán los contadores asíncronos? La solución la tienen los profesionales que con su uso o desuso animan a los fabricantes en un sentido u otro. Actualmente, todo parece indicar que los contadores asíncronos dejarán de ser fabricados.

9.4. Comparación asíncrono vs síncrono

En este apartado ordenaremos las diferencias entre ambas estrategias con el fin de favorecer conclusiones más claras.

Primeramente recordemos que en un contador asíncrono el valor de cada bit es función del valor del bit anterior en el mismo flanco del reloj, mientras que en un contador síncrono el valor de cada bit es función del valor de los anteriores bits en el flanco anterior. Reescrito lo anterior:

Asíncrono	Síncrono
$Q_i(t) = f(Q_{i-1}(t))$	$Q_i(t)=f(Q_0, Q_1,..,Q_{i-1}(t-1))$
$Ck_i = Q_{i-1}(t)$	$Ck_i = Ck$
$J_i = K_i = 1$	$J_i = K_i = Q_0 \cdot Q_1 \cdot ... \cdot Q_{i-1}(t-1)$

De las anteriores expresiones y circuitos:

- El primer bit siempre bascula, tanto para síncronos como para asíncronos.
- En el síncrono el estado t se genera para todos los bits a la vez a partir del estado t-1.
- En el asíncrono el estado se genera bit a bit.
- En el síncrono el circuito controla el qué se hace (valores de J y K) y no el cuándo se hace (Ck fijo para todos los J-K).
- En el asíncrono el circuito lógico controla para cada J-K cuándo lo hace (señal de Ck de cada J-K) y no el qué hace (J y K siempre a 1).
- En el asíncrono el paso entre ciertos estados genera glitches.
- El síncrono no presenta glitches entre estados.

La tabla 9-7 ordena las diferencias entre contadores síncronos y asíncronos.

Tabla 9-7
Comparación entre estrategias de diseño.

	Contador Asíncrono	Contador Síncrono	Contador con Acarreo
Tiempo Evol.	n x tpd	tpd + tand	tpd+(n-2)tand
Rapidez	3º	1º	2º
Frec. máxima	$\frac{1}{n \times tpd}$	$\frac{1}{tpd+tand}$	
Sencillez	1º	2º	3º
Ampliación	SÍ	SÍ	SÍ
Implement.	SÍ	NO	SÍ
Ind. flanco	NO	SÍ	SÍ
Glitch	SÍ	NO	NO
Uso	Desuso	Activo	Activo

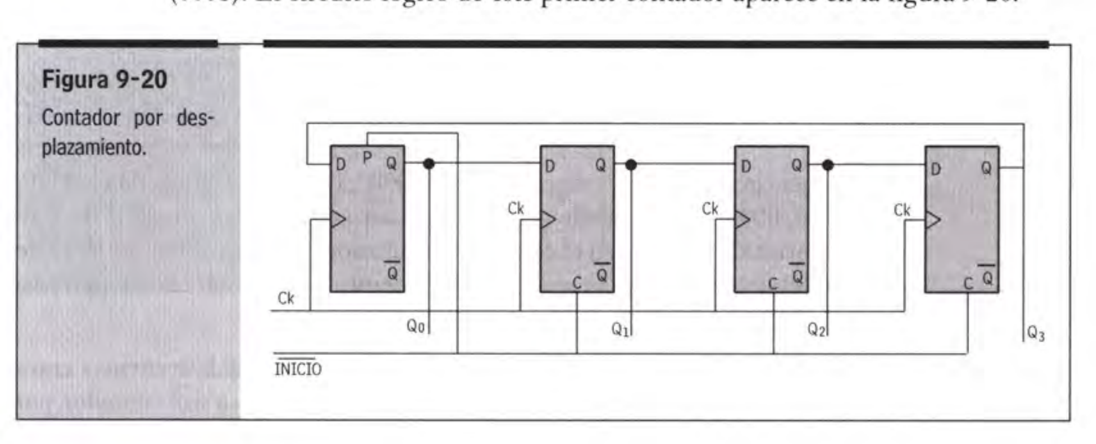
9.5. Otros contadores

En los apartados anteriores hemos visto cómo diseñar un contador de módulo completo (2, 4, 8, 16, etc.) en binario puro. Faltan por diseñar otros contadores, como el BCD (módulo 10) o el de código Johnson.

9.5.1. Contador Johnson

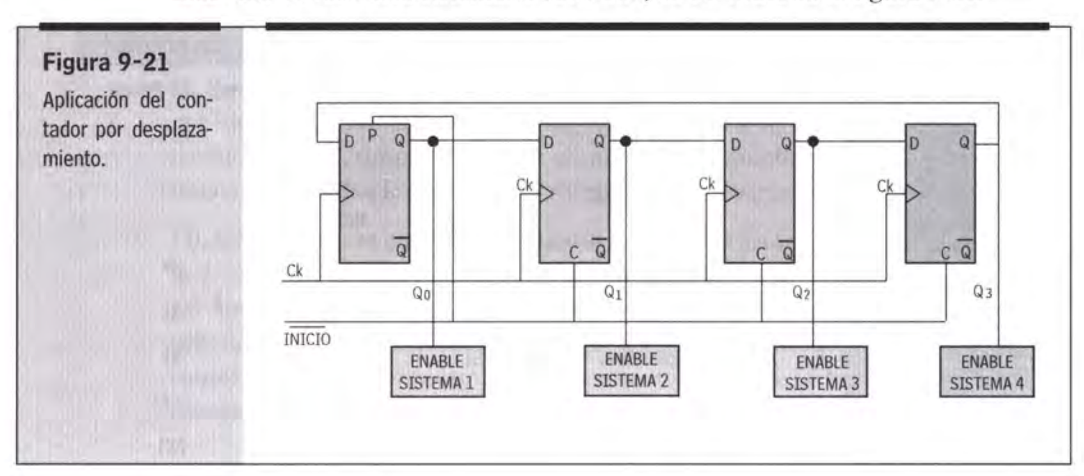
Antes del contador en código Johnson veamos su predecesor. La secuencia a generar es para cinco bits: 0001, 0010, 0100, 1000, 0001, etc. Si nos fijamos, la secuencia se obtiene por simple desplazamiento tras una correcta inicialización (0001). El circuito lógico de este primer contador aparece en la figura 9-20.

Figura 9-20
Contador por desplazamiento.



Veamos una aplicación de este contador. Imaginemos que Ck tuviera una frecuencia de 1 Hz, y que conectáramos cada salida del contador al ENABLE de cada uno de los cinco sistemas a controlar, como muestra la figura 9-21.

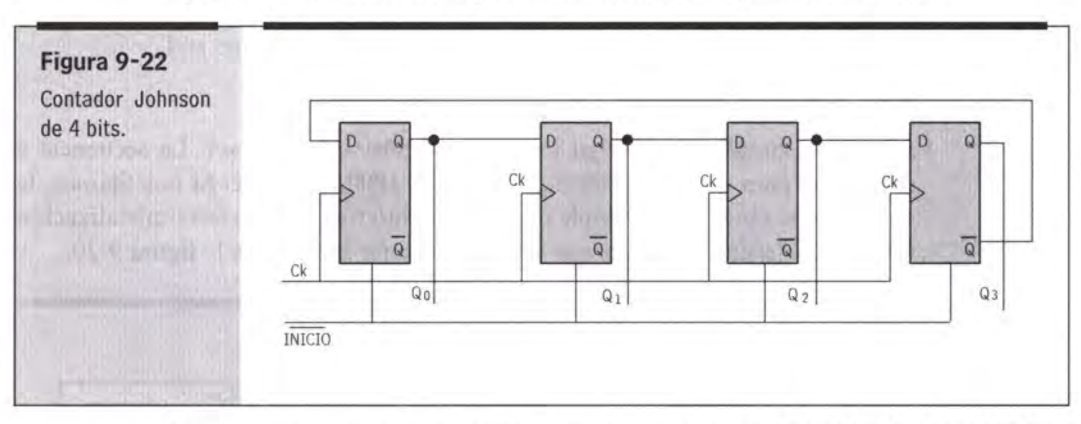
Figura 9-21
Aplicación del contador por desplazamiento.



Cada sistema se conectaría un segundo de cada cinco segundos. Éste es un caso típico en el que el contador establece un control por secuencia.

El contador de Johnson modifica ligeramente el circuito y bastante la secuencia. Simplemente recirculará a D0 el valor negado de Q4, en vez del propio Q4. La figura 9-22 es el circuito lógico de un contador de Johnson de 4 bits.

Figura 9-22
Contador Johnson de 4 bits.



La secuencia generada por este contador es: 0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000, 0000, etc. Se puede ver que los cuatro estados del contador de 9-20 se han convertido en ocho en el contador de Johnson, es decir, el rendimiento de los bits es el doble, aunque la secuencia del circuito 9-20 es más clara e intuitiva que la de Johnson.

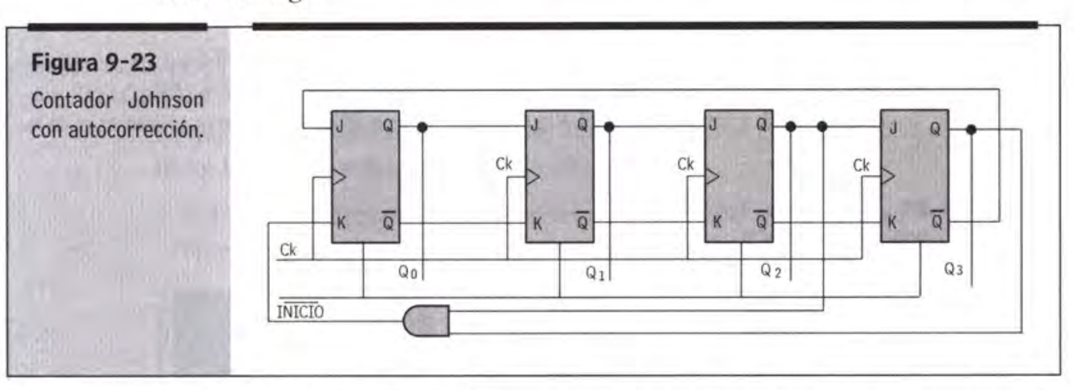
En ambos casos queda una pregunta en el aire: ¿qué pasa si la secuencia entra en un estado espurio no contemplado? Por ejemplo, ¿qué pasa si el contador, por la razón que sea, toma el valor 0010? La respuesta puede ser una de estas dos:

- La secuencia vuelve a la normalidad en un tiempo finito.
- La secuencia queda corrupta para siempre; no vuelve a la normalidad.

En el caso del contador de Johnson la respuesta es evidente: el contador no volverá a la normalidad; la secuencia quedará corrupta hasta que se active INICIO.

Una solución a este problema es modificar ligeramente el circuito de modo que si la secuencia se corrompiera, ésta volvería a la normalidad en un número de flancos finito. Por ejemplo, el contador de Johnson modificado de la figura 9-23 se autocorrige.

Figura 9-23
Contador Johnson con autocorrección.



Analicemos en la tabla 9-8 la evolución del contador Johnson a partir del estado espurio 0010. Pasados tres flancos el contador recupera la normalidad. El planteamiento de la figura 9-23 puede ser extendido a cualquier número de bits, sin más que asignar a K0 el valor de $Q_n \cdot Q_{n-1}$.

Tabla 9-8
Evolución de un contador Johnson.

t						t+1			
Q3	Q2	Q1	Q0	J0	K0	Q3	Q2	Q1	Q0
0	0	1	0	1	0	0	1	0	1
0	1	0	1	1	0	1	0	1	1
1	0	1	1	0	0	0	1	1	1
0	1	1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1	1	0
1	1	1	0	0	1	1	1	0	0
..	..	..	..	..	..	..	..	..	..

9.5.2. Diseño de cualquier contador asíncrono

Lo primero que hay que señalar es que si bien se puede diseñar asíncronamente cualquier secuencia en cualquier código, sólo tiene sentido hablar de secuencias ordenadas codificadas en binario puro, por ejemplo de 0 a 9, de 0 a 11 o de 0 a 59.

Los pasos a dar para diseñar un contador asíncrono de módulo inferior al posible son:

1. Obtener el circuito asíncrono correspondiente al módulo completo.
2. Decodificar mediante una puerta NAND el primer estado no deseado.
3. Conectar la salida de la NAND a las entradas Preset y Clear que fuercen en los biestables el primer estado válido de la secuencia.

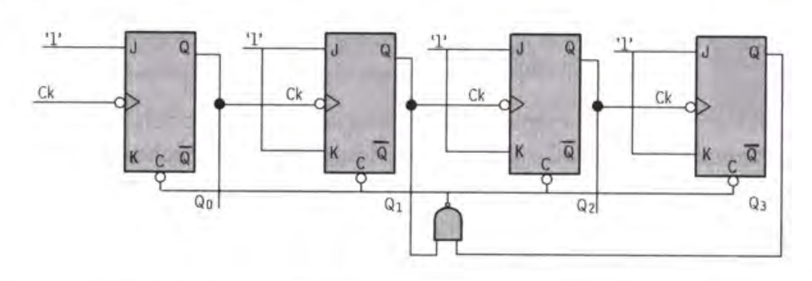
Ejemplo 9-1

Diseño de un contador BCD asíncrono.

Para diseñar un contador BCD de 0 a 9, el primer estado no deseado será el 1010 y el estado a restaurar será el 0000. Por tanto, la puerta NAND recibirá en sus entradas a Q3, $\overline{Q2}$, Q1 y $\overline{Q0}$, o sólo a Q3 y Q1 previa simplificación, y su salida se conectará a la CLEAR de los biestables. El circuito resultante es el de la figura 9-24.

Figura 9-24

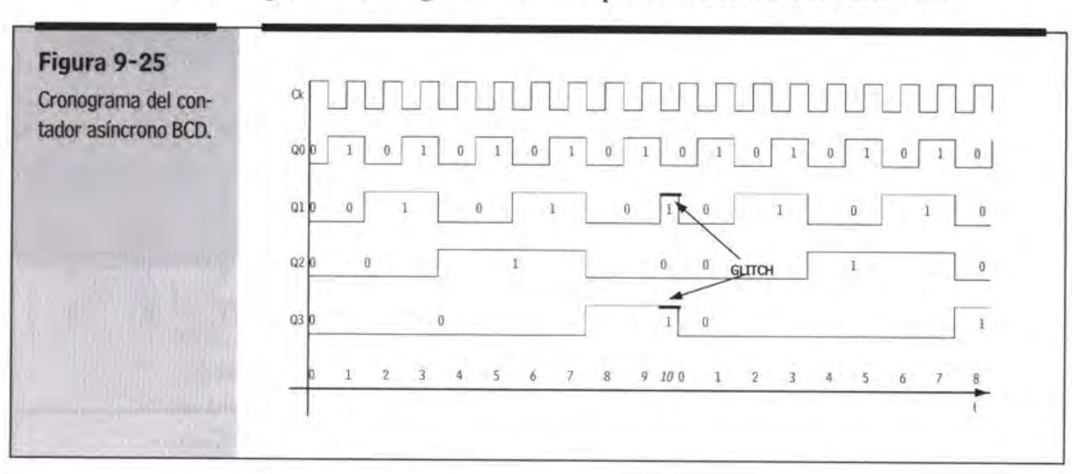
Contador asíncrono BCD.



El cronograma de la figura 9-25 es el que se obtiene del circuito 9-24.

Figura 9-25

Cronograma del contador asíncrono BCD.



Como se ha destacado exageradamente en el cronograma, el contador pasa por el estado 1010 durante el tiempo de actuación de la puerta NAND y del CLEAR, generándose un glitch que puede dar problemas en otras partes del sistema. Además, si nos fijamos en el paso de 1010 a 0000, en la señal Q1 se ha generado

un flanco descendente, lo que haría bascular a Q2, o no, según fueran los tiempos de evolución de la NAND y los J-K. Es decir, no pasaría de 1010 a 0000, sino a 0100. Así pues, este circuito no es práctico, aunque sí tiene valor teórico.

El planteamiento seguido para el contador BCD puede ser aplicado a cualquier otro contador, pero en estos casos lo más normal es usar los contadores MSI del punto 9.6.

9.5.3. Diseño de cualquier contador síncrono

La estrategia síncrona permite diseñar un contador que genere cualquier secuencia en cualquier código. Esta implementación será siempre válida, sin glitches y sin los problemas y limitaciones de los contadores asíncronos.

El método se aproxima al diseño de autómatas que veremos en el capítulo 10 del libro. Los pasos a dar son los siguientes:

1. Dibujar el diagrama con tantos estados (círculos) como tenga la secuencia a obtener, y relacionarlos en orden con las transiciones (flechas).
2. Asignar a cada estado la codificación elegida libremente (cualquier código).
3. Escribir la tabla de verdad con el estado t en la entrada y el t+1 en la salida, teniendo en cuenta el diagrama de los puntos 1 y 2.
4. Añadir a la tabla de verdad las entradas D de los biestables. Cada columna D tomará el valor de Qt+1.
5. Simplificar las señales D.
6. La secuencia del contador está en las salidas de los biestables.
7. Añadir la línea de INICIO.

Apliquemos los pasos anteriores diseñando un contador BCD síncrono.

Diseñemos el contador BCD síncrono ascendente. El diagrama del contador de los puntos 1 y 2 es el de la figura 9-26.

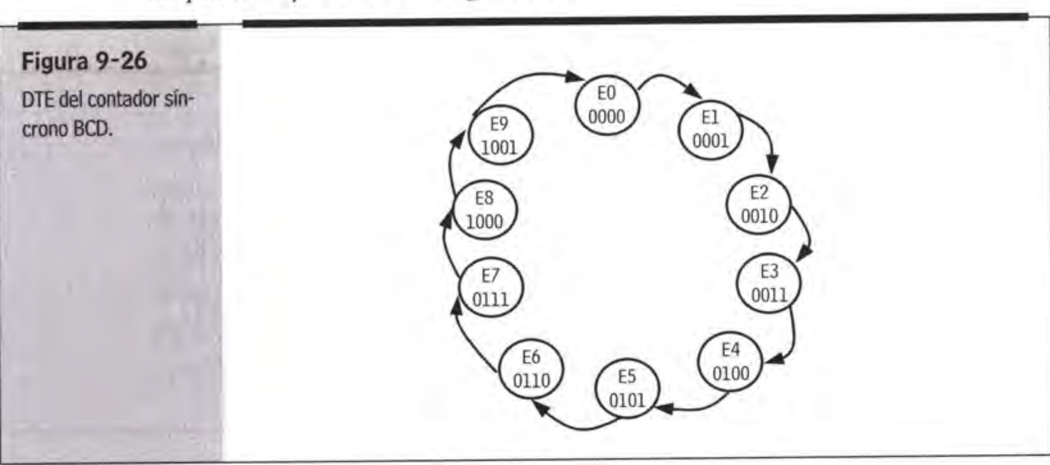


Figura 9-26
DTE del contador síncrono BCD.

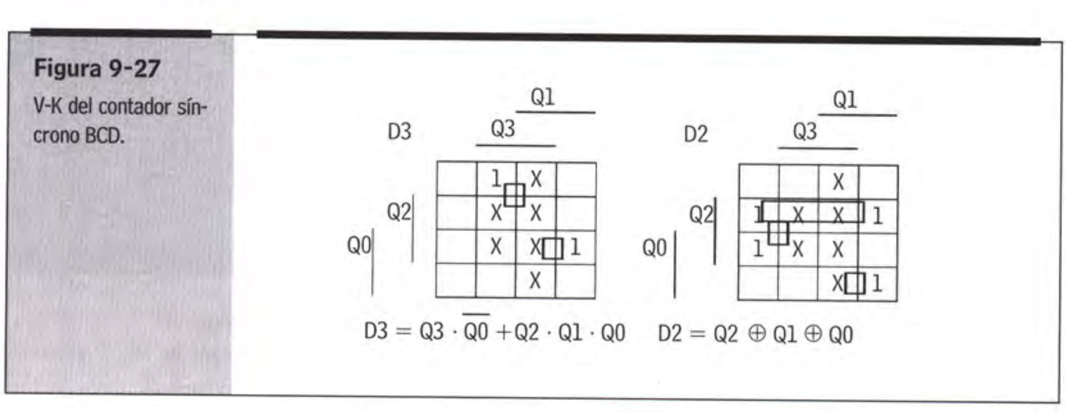
Queda claro que las flechas podrían haber relacionado los estados en cualquier orden (E3, E2, E7, E1,..), y que podríamos haber codificado cada estado en cualquier código (XS3, Aiken, etc.). La tabla de verdad correspondiente a la figura 9-26 es la tabla 9-9.

Tabla 9-9
T-V de un contador BCD síncrono.

t				t+1							
Q3	Q2	Q1	Q0	Q3	Q2	Q1	Q0	D3	D2	D1	D0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	1	0	0	1	1
0	0	1	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1
0	1	1	1	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1	0	0	1
1	0	0	1	0	0	0	0	0	0	0	0
1	0	1	0	X	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X

Los diagramas de V-K y su correspondiente simplificación aparecen en la figura 9-27.

Figura 9-27
V-K del contador síncrono BCD.



BIBLIOTECA CADIZ PUBLICA

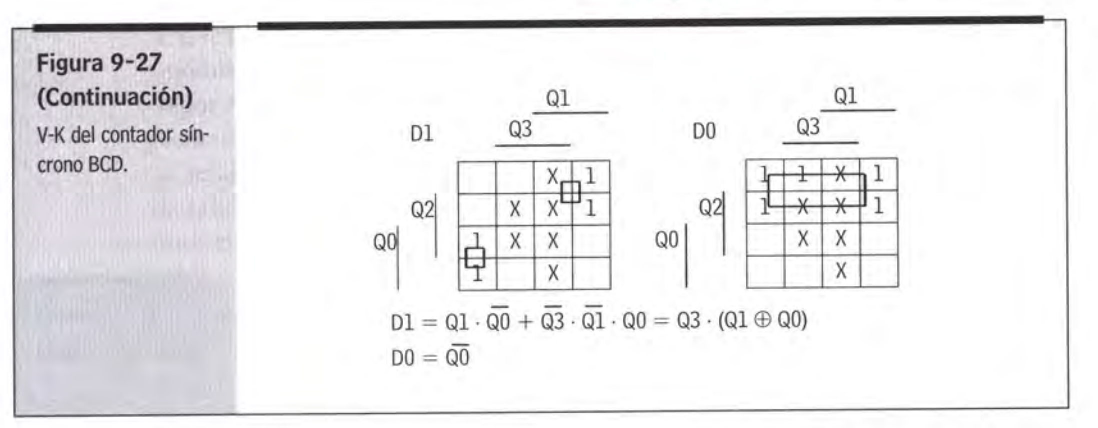


Figura 9-27 (Continuación)
V-K del contador síncrono BCD.

El circuito lógico correspondiente a las expresiones simplificadas en 9-27 aparece en la figura 9-28.

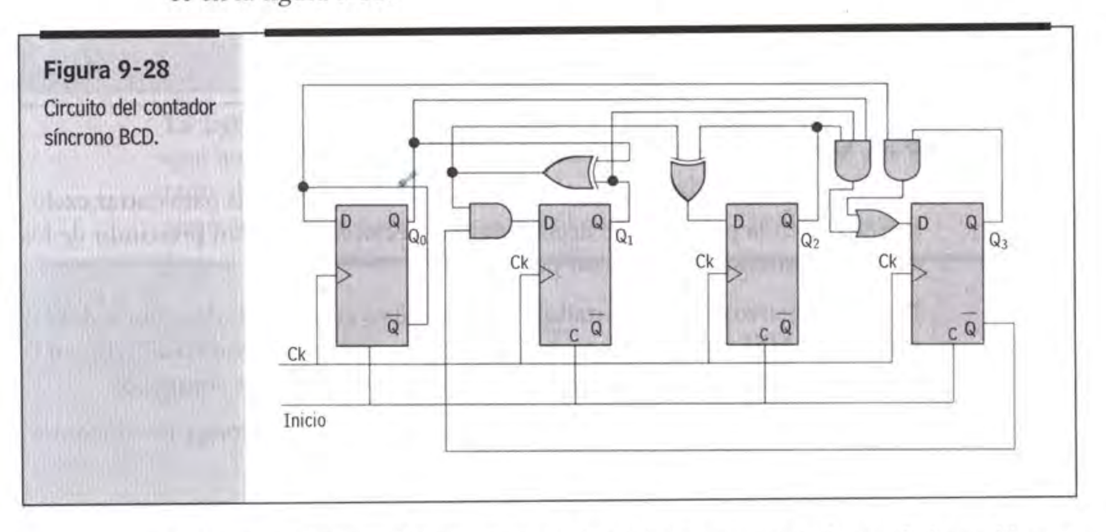


Figura 9-28
Circuito del contador síncrono BCD.

Volvamos a los estados espurios. En la tabla hemos asociado condiciones libres a los estados no posibles (1010-1111), de tal forma que si por ruido u otra causa extraña se diera el estado 1011, ¿cuál sería el siguiente estado?, ¿volvería la secuencia a la normalidad? Para saber lo que el contador hace hay que obtener el cronograma en detalle o analizar las expresiones de D, o mejor aún, utilizar un programa de simulación que facilite el estudio. La solución más elegante y preferible pasa por no considerar ningún estado como imposible, desapareciendo las X. Por ejemplo, la figura 9-29 es el diagrama que contempla todos los estados y que lleva a los posibles estados espurios a un estado conocido, el 0000, que podría haber sido cualquier otro.

En este caso, cada vez que el contador pase por un estado espurio, volverá en el siguiente flanco activo al estado E0, recuperando la normalidad.

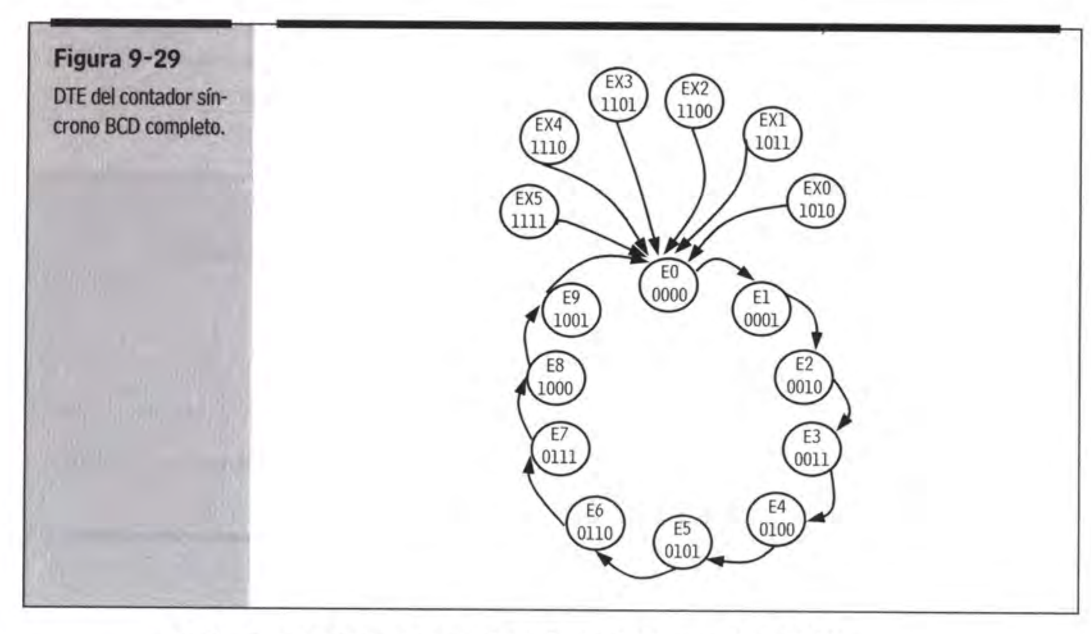


Figura 9-29
DTE del contador síncrono BCD completo.

9.6. Contadores en circuitos integrados MSI

En este apartado abandonaremos las consideraciones teóricas para entrar exclusivamente en la práctica. De hecho, algunos lectores preferirán prescindir de los apartados anteriores y empezar por éste.

No es el objetivo de este apartado (ni de este libro en general) describir todos los contadores MSI disponibles, sino aquellos que por básicos ayuden al lector en la mayoría de los casos, y le preparen para abordar diseños más complejos.

Describiremos primeramente los contadores asíncronos y luego los síncronos, pudiéndose empezar la lectura por cualquiera de ellos.

9.6.1. Contadores asíncronos

Los principales y más sencillos contadores asíncronos son el 7490 y el 7493.

9.6.1.1. 7490: Contador asíncrono BCD

El esquema funcional de un contador asíncrono ascendente BCD activo por flancos descendentes aparece en la figura 9-30.

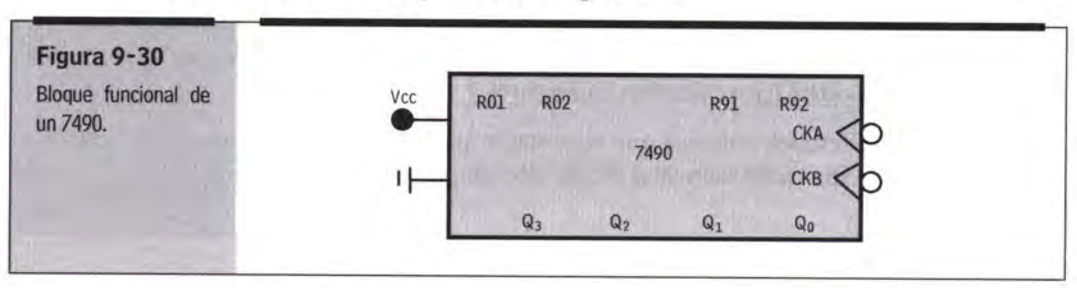


Figura 9-30
Bloque funcional de un 7490.

En realidad, el 7490 no tiene un contador, sino dos: uno de módulo 2 y otro de módulo 5. Por eso hay dos relojes, CKA y CKB. La señal CKA es el reloj del contador módulo 2, y CKB es el reloj del contador de módulo 5. Para obtener un contador de módulo 10 sólo hay que unir la salida del contador módulo 2 (Q0) al reloj del contador módulo 5 (CKB), siguiendo la estrategia asíncrona de que la salida de un biestable es el reloj del siguiente. Aplicando esta idea obtendremos el circuito de la figura 9.31.

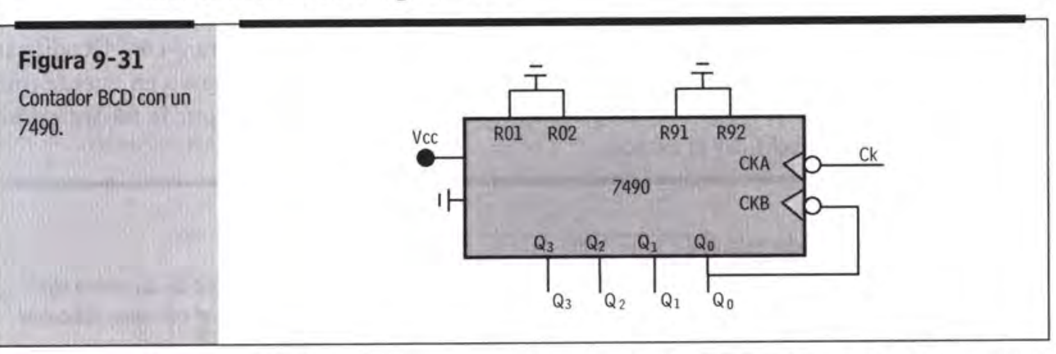


Figura 9-31
Contador BCD con un 7490.

La figura 9-32 muestra un contador asíncrono módulo 10 en un código BCD que no es el puro, simplemente se han intercambiado los Ck y se ha reinterpretado el orden de las salidas. Queda como trabajo para el lector determinar la secuencia que genera el circuito y el código BCD de que se trata.

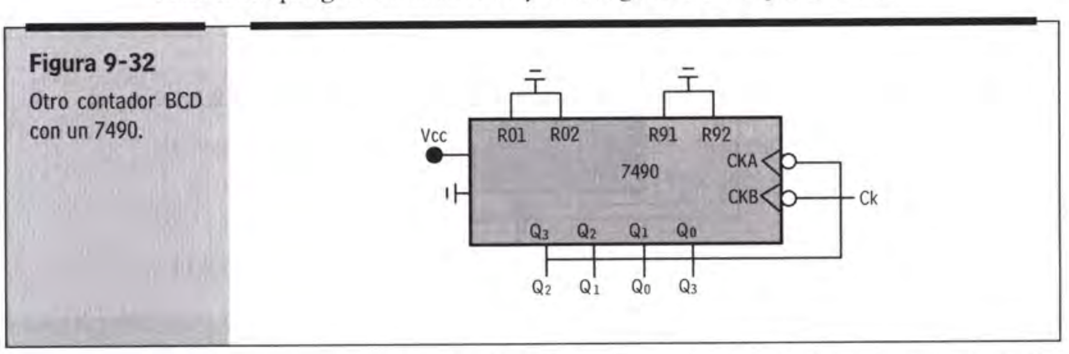


Figura 9-32
Otro contador BCD con un 7490.

Estudiemos con más detalle el 7490 mediante la tabla de funcionamiento 9-10.

Tabla 9-10
Tabla del contador 7490.

Ck	R01	R02	R91	R92	Q3	Q2	Q1	Q0	
X	1	1	0	0	0	0	0	0	0
X	0	0	1	1	1	0	0	1	9
↓	0	X	0	X		$Q_{3\text{-}0}$ (t - 1) +1			+1
↓	0	X	X	0		$Q_{3\text{-}0}$ (t - 1) +1			+1
↓	X	0	0	X		$Q_{3\text{-}0}$ (t - 1) +1			+1
↓	X	0	X	0		$Q_{3\text{-}0}$ (t - 1) +1			+1

Para obtener un contador *normal* basta con poner R01, R02, R91 y R92 a tierra. De este modo, el 7490 contará de 1 en 1 por cada flanco descendente.

Las líneas R01, R02, R91 y R92 son de puesta a 0 y a 9 asíncrona, respectivamente. Si R01 y R02 valieran 1, de forma inmediata el contador pasaría al estado 0000 (0); este forzado no necesita de flancos, es asíncrono. Lo mismo para R91 y R92, pero forzando el valor 9.

Las líneas de puesta a 9 se usan poco, no así las de puesta a 0, que valen para diseñar contadores de módulo inferior a 9. En este caso bastará con decodificar el primer estado no deseado y conectarlo a R01 y R02, teniendo en cuenta que R0 y R02 se unen en una puerta AND interna al 7490, que puede ser suficiente para decodificar el estado.

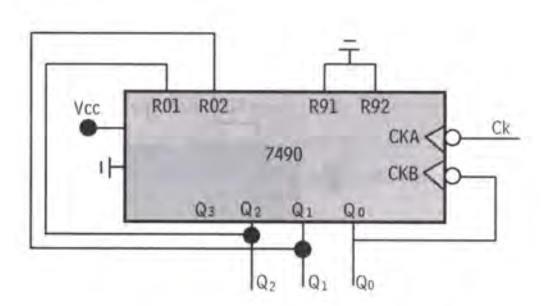
Ejemplo 9-2

Diseñar un contador módulo 6 con un 7490.

Al diseñar un contador de módulo 6 (0-5) el primer estado no deseado es el 110 (6), que se decodifica directamente con Q2 y Q1 y con la puerta AND interna del 7490. La figura 9-33 muestra al contador asíncrono módulo 6.

Figura 9-33

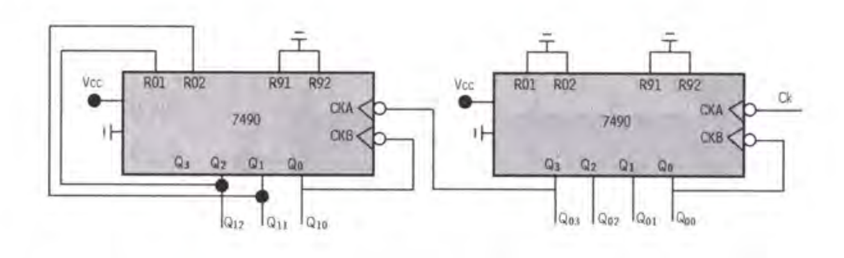
Contador módulo 6 con un 7490.



Podemos diseñar también un contador de segundos (módulo 60), uniendo un contador módulo 10 con otro de módulo 6, obteniendo el circuito de la figura 9-34.

Figura 9-34

Contador módulo 60 con un 7490.

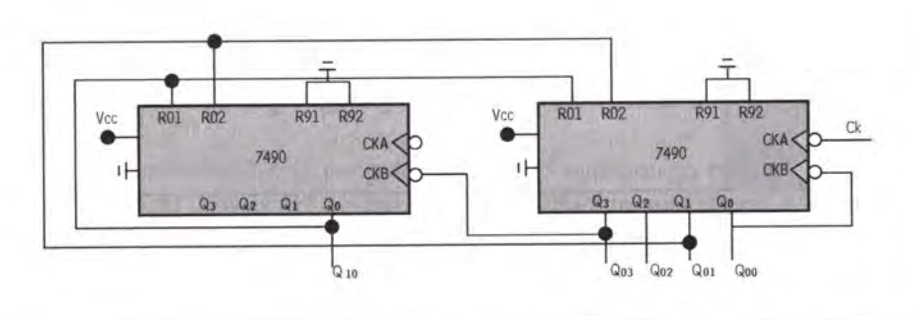


Ejemplo 9-2
(Continuación)

Cuando el primer 7490 pase del estado 9 al 0, el bit Q3 pasará de 1 a 0, generando un flanco descendente que por estar conectado al reloj del siguiente 7490 hará que éste incremente en 1 su valor.

Si replicáramos el circuito 9-34 y lo conectáramos con él, obtendríamos un contador de segundos y minutos; casi un reloj. Nos faltaría diseñar un contador de horas; éste es más complejo, pues su módulo 12 es mayor que 10. En este caso la secuencia a generar es 0 0000 (00), 0 0001 (01), 0 0010 (02), ...0 1001 (09), 1 0000 (10), 1 0001 (11), 0 0000 (00), etc. El circuito de la figura 9-35 implementa un contador asíncrono de módulo 12.

Figura 9-35
Contador módulo 12 con 7490



9.6.1.2. 7493: Contador asíncrono binario

El 7493 es un contador asíncrono ascendente de módulo 16 activo por flanco descendente. No es ni mejor ni peor que el 7490, simplemente aquél cuenta en BCD y éste lo hace en binario puro.

El bloque funcional del 7493 aparece en la figura 9-36.

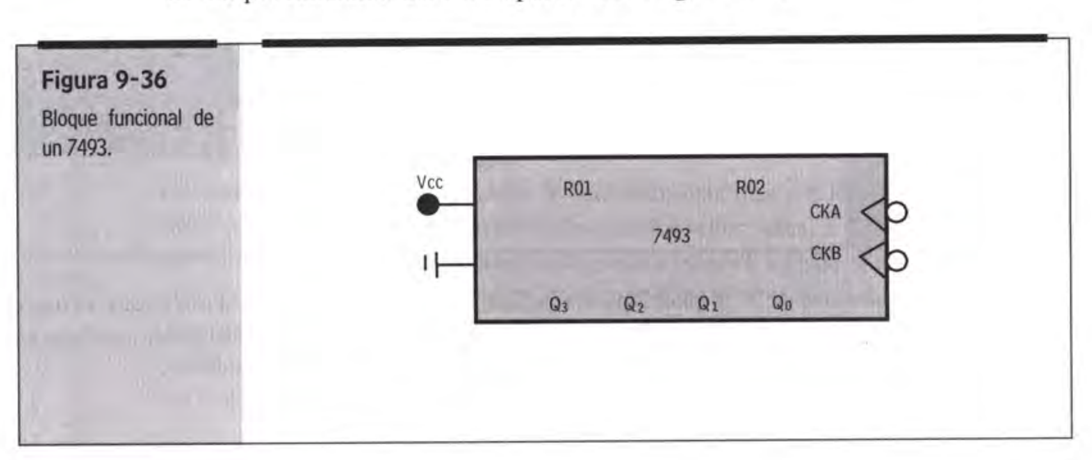


Figura 9-36
Bloque funcional de un 7493.

El 7493 dispone de dos contadores, uno de módulo 2 y otro de módulo 8. De esta manera el 7493 puede ser utilizado como dos contadores (de módulo 2 y 8) o como un contador de módulo 16. Cabe decir que el contador de módulo 2 es muy utilizado en la práctica. La figura 9-37 presenta un contador asíncrono módulo 16.

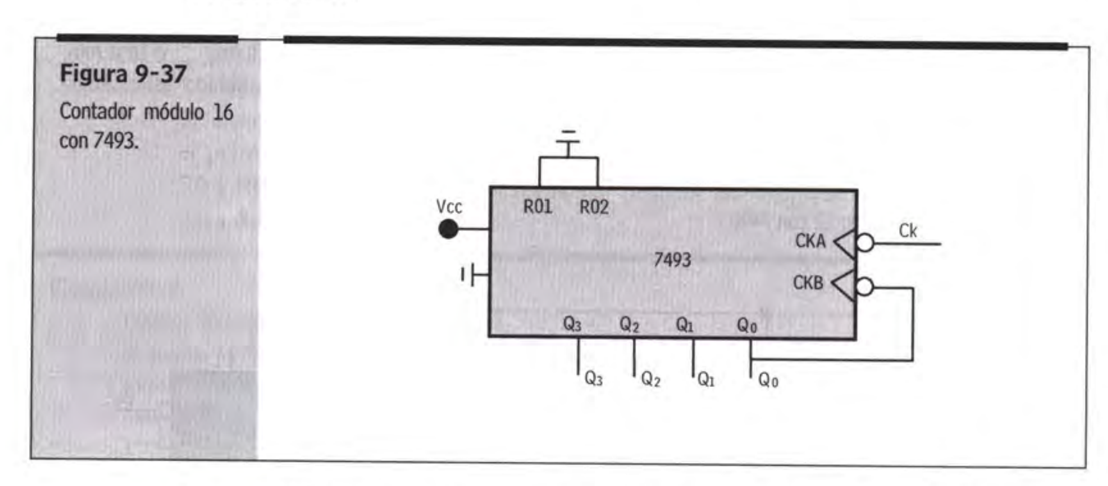


Figura 9-37
Contador módulo 16 con 7493.

La tabla de funcionamiento es casi idéntica a la del 7490, sólo que el 7493 no dispone de R91 y R92. Utilizando las líneas R01 y R02 del 7493 podemos implementar un contador BCD como el de la figura 9-38.

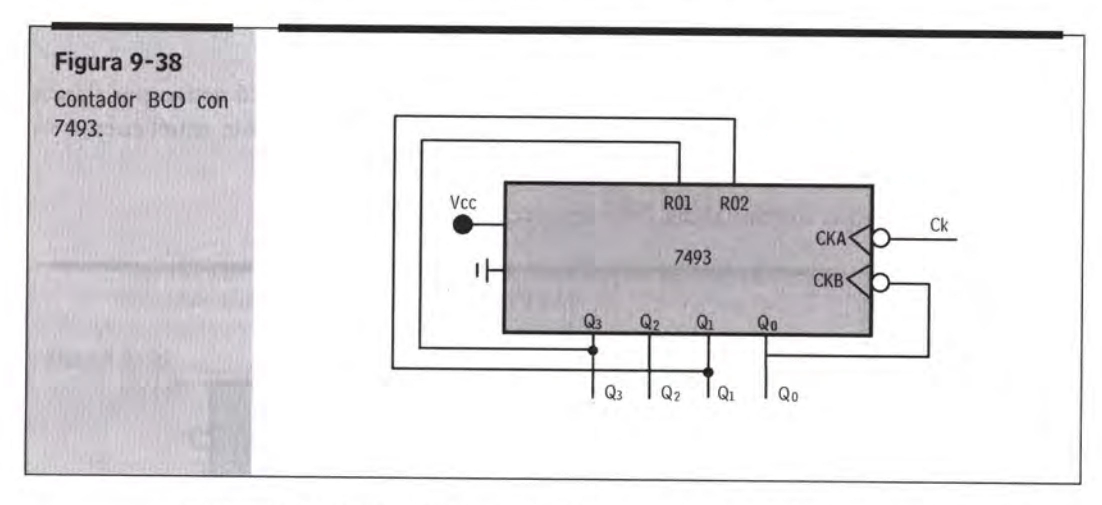


Figura 9-38
Contador BCD con 7493.

Al circuito de 9-38 podríamos añadirle una línea de inicialización externa como en 9-39. Así, el contador pasaría a 0000 tanto al llegar a 9 como al ser activado el pulsador de RESET.

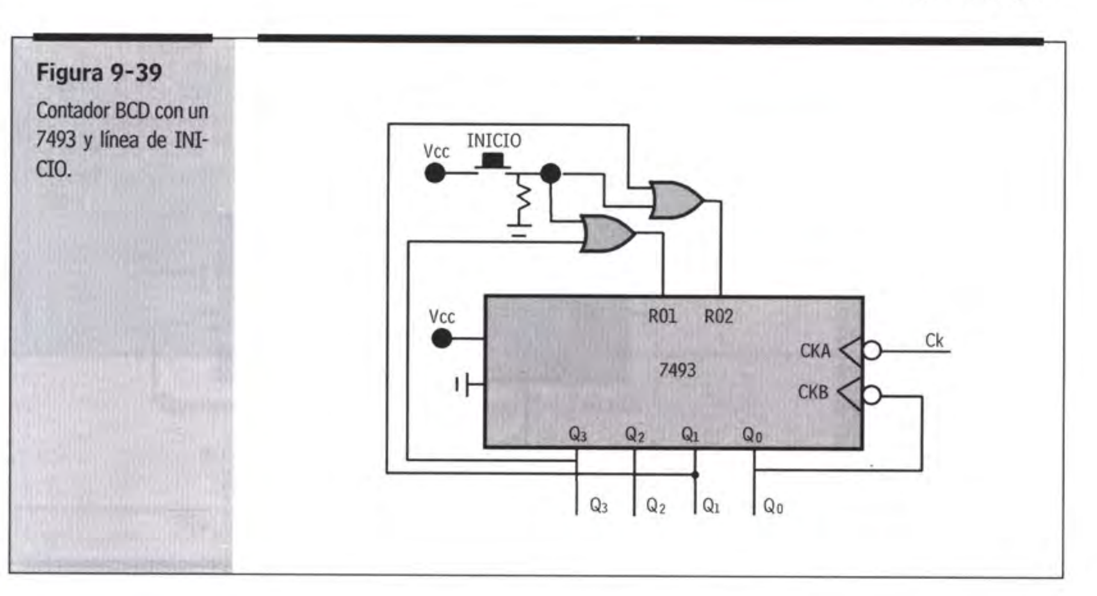


Figura 9-39
Contador BCD con un 7493 y línea de INICIO.

O podríamos diseñar un contador módulo 13 teniendo en cuenta que el estado a decodificar sería el 1101, y por tanto nos haría falta una puerta AND externa, como muestra la figura 9-40.

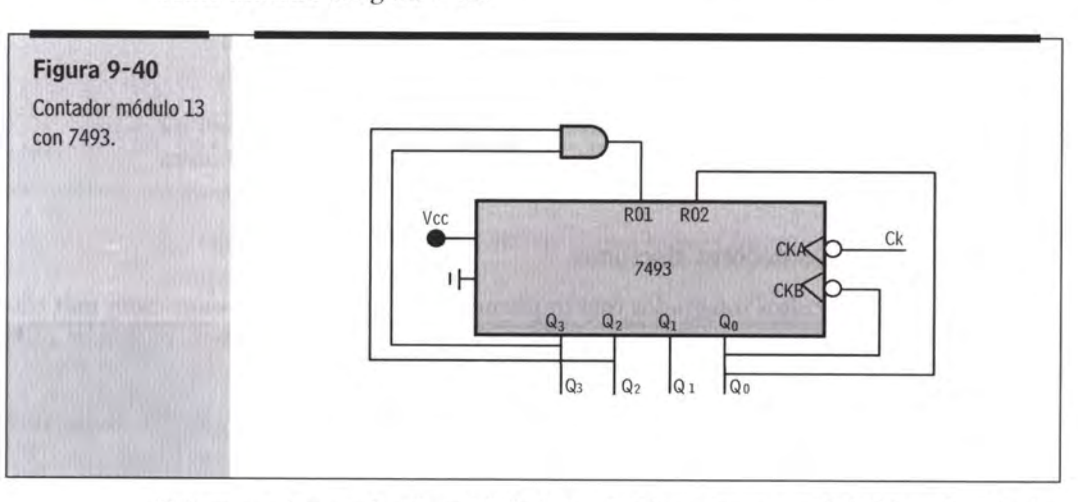


Figura 9-40
Contador módulo 13 con 7493.

Volvamos al contador de módulo 16 y añadámosle una red RC. Este simple circuito con una simple resistencia y un condensador (p.ej. 1 K y 1 μF) asegura que el contador al ser conectado a la red se pondrá a 0000. Es decir, siempre que haya *power on* el contador se reinicializará por sí mismo, empezando siempre desde 0000 (lo que no tiene por qué ser siempre una ventaja). Además, se puede añadir una línea de INICIO externa. Las figuras 9-41 y 9-42 resumen lo dicho.

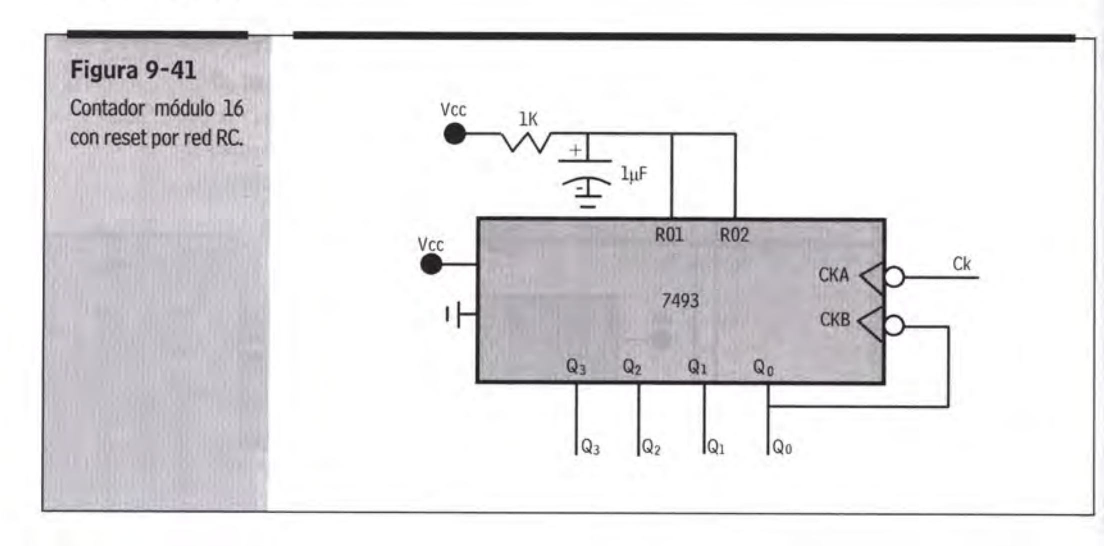


Figura 9-41
Contador módulo 16 con reset por red RC.

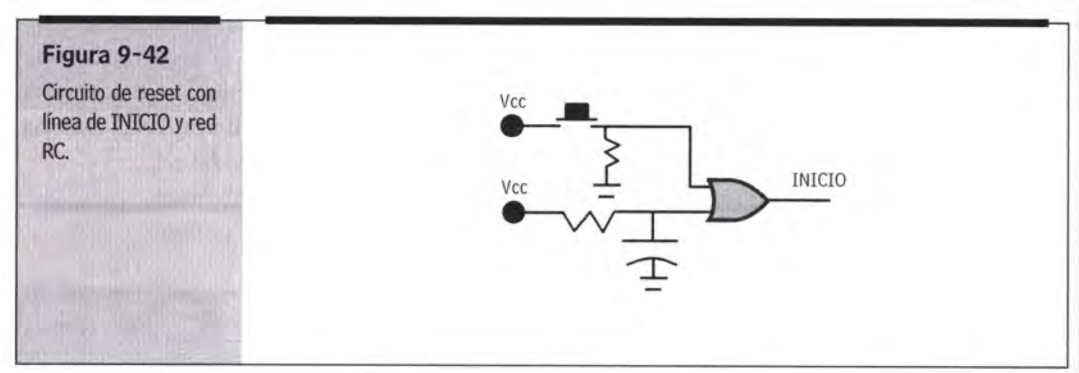


Figura 9-42
Circuito de reset con línea de INICIO y red RC.

9.6.2. Contadores síncronos

Los circuitos integrados que implementan contadores síncronos están más elaborados que los asíncronos, ofrecen al usuario más funciones, relegando cada vez más a los asíncronos.

A continuación describiremos los contadores 74160, 74163 y 74190. Todos ellos disponen de reloj y salidas, y de varias líneas auxiliares:

- Clear: pone el contador a 0.
- Load: fuerza el valor del contador a un valor externo.
- Enable: habilita o inhabilita el contador.
- Tope: indica si el contador está en el último estado.

9.6.2.1. 74160: Contador síncrono BCD

El 74160 es un contador síncrono ascendente BCD activo por flancos ascendentes. La figura 9-43 muestra el bloque funcional del 74160.

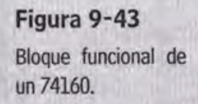
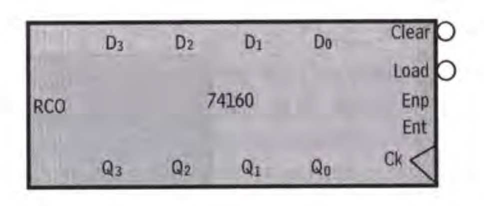


Figura 9-43
Bloque funcional de un 74160.



Resumamos el comportamiento del 74160:

- Cuenta al ritmo de los flancos ascendentes del reloj según el módulo 10 (0000 a 1001).
- Si se activara CLEAR el contador se pondría a 0 al llegar el siguiente flanco ascendente, es decir, síncronamente.
- Si se activara LOAD el contador se cargaría en el siguiente flanco ascendente con D3-0.
- El contador cuenta sólo si las líneas ENP y ENT están a 1; si estuviera una sola de ellas a 0 el 74160 no contaría. ENP y ENT son distintas.
- RCO se pondrá a 1 (se activa) si la salida Q3-0 alcanza su último valor, en este caso el 9. RCO=1001 · ENT (sin ENP)

La línea CLEAR vale para reiniciar. Las líneas RCO, ENP y ENT sirven para extender el módulo del contador, mientras que las líneas LOAD y D3-0 sirven para diseñar contadores de módulo inferior a 10.

La tabla funcional 9-11 describe de forma breve (aunque no muy completa) el comportamiento del 74160.

Tabla 9-11
Tabla del contador 74160.

$\overline{\text{CLR}}$	$\overline{\text{LOAD}}$	ENP	ENT	Ck	Q3-0	RCO
0	X	X	X	X	0000	0
X	0	X	X	X	D3-0	0
1	1	1	1	↑	Cuenta	0
1	1	1	1	↑	1001	1
1	1	0	1	X	No cuenta	0
1	1	1	0	X	No cuenta	0

El contador módulo 6 de la figura 9-44 se obtiene decodificando con una NAND el último estado deseado 5=0101 (y no el primer no deseado como en los asíncronos), y conectándolo a LOAD. Además, en D3-0 forzaremos el valor de reinicio de la cuenta, en este caso 0000.

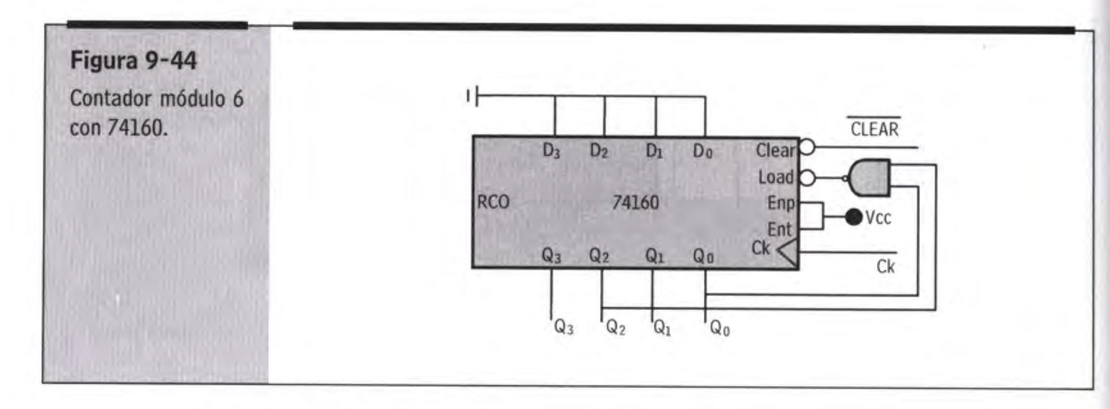


Figura 9-44
Contador módulo 6 con 74160.

La figura 9-45 implementa un contador de 3 a 8, secuencia que no era posible implementar con un 7490.

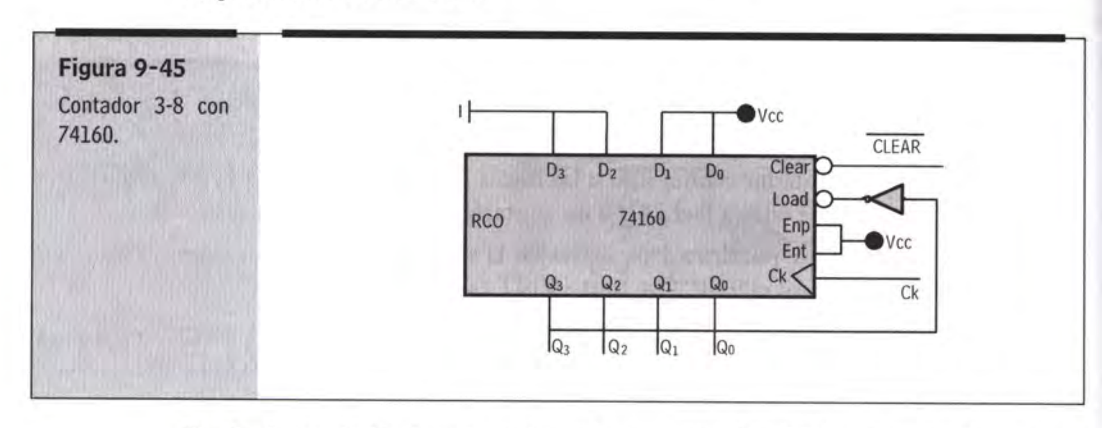


Figura 9-45
Contador 3-8 con 74160.

En la figura 9-45, la reinicialización con el valor 0011 no se produce en el momento en que Q3 pasa a 1, ya que la carga con LOAD es síncrona. Es decir, el estado 1000 estará presente hasta la llegada del siguiente flanco ascendente, que será el que reinicialice el contador.

Para unir varios contadores con el fin de obtener módulos superiores a 10 hay que utilizar las líneas ENP, ENT y RCO. Por ejemplo, la figura 9-46 es un contador de módulo 60 implementado con dos 74160.

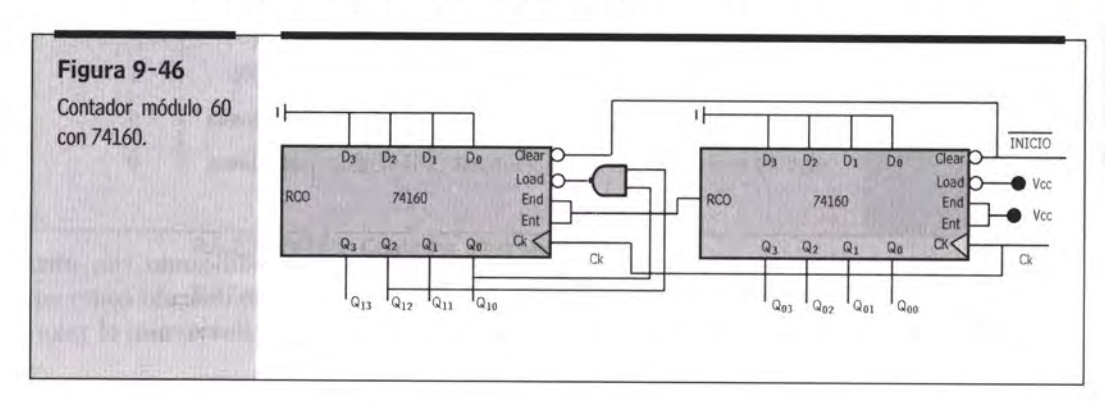


Figura 9-46
Contador módulo 60 con 74160.

En la figura 9-46, mientras el primer 74160 no alcance el valor 9 su salida RCO será 0 y por tanto el segundo 74160 no contará, aunque reciba flancos en el Ck. Sin embargo, cuando el primer 74160 alcance el valor 9 RCO pasará a 1, quedando habilitado el segundo 74160, de modo que cuando llegue el siguiente flanco del reloj, el primer 74160 pasará a 0 y el segundo aumentará en 1 su valor. Fijémonos en que el segundo 74160 no aumenta su valor cuando el primero está a 9, ya que para cuando RCO se ha puesto a 1 ya ha desaparecido el flanco del reloj.

Al anterior esquema también se le pueden añadir líneas de carga externa y sobre todo la red RC vista en los contadores asíncronos.

9.6.2.2. 74163: Contador síncrono binario

El 74163 es idéntico al 7493 (contador binario) en cuanto a la secuencia generada, y al 74160 (síncrono) en cuanto a su extensión, así que no abundaremos mucho en él. La figura 9-47 muestra un contador de módulo 1.024 implementado con 74163.

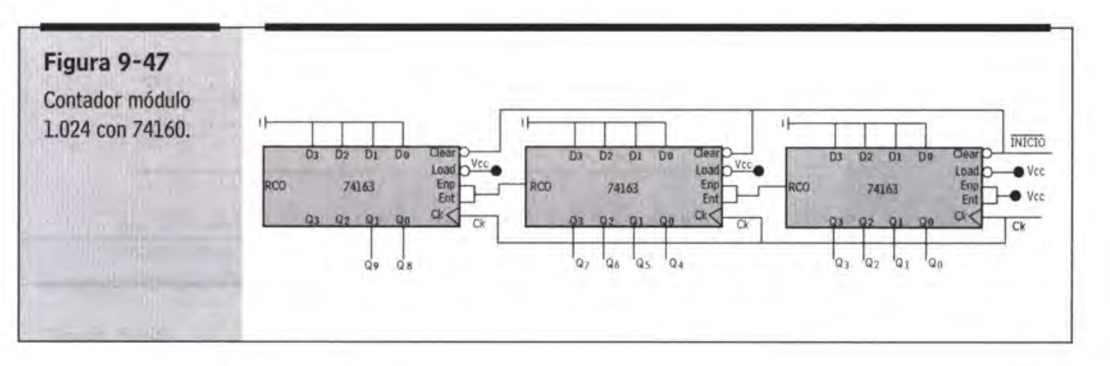


Figura 9-47
Contador módulo 1.024 con 74160.

9.6.2.3. 74190: Contador BCD ascendente/descendente

Este contador dispone de una línea U/D; si su valor fuera 0 el contador sería ascendente al ritmo de los flancos ascendentes del reloj, mientras que si U/D fuera 1 contaría descendentemente. El bloque del 74190 aparece en la figura 9-48.

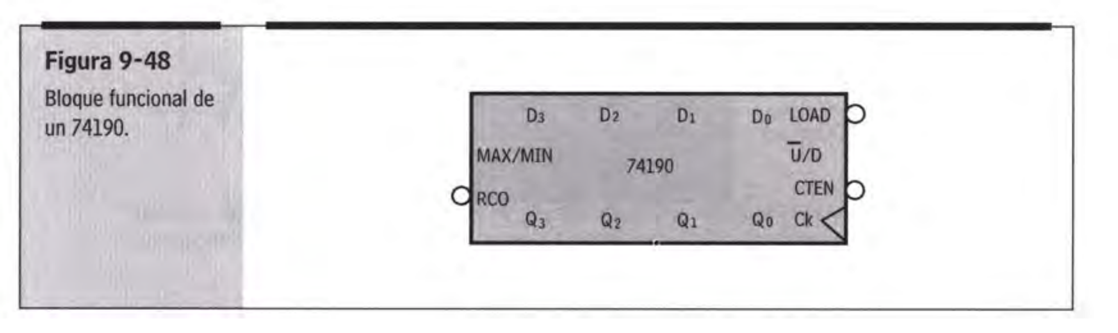


Figura 9-48
Bloque funcional de un 74190.

Vemos que ha desparecido la línea CLEAR, ya que ahora la reinicialización podrá ser a 0 o a 9. Además, vemos dos líneas, MAX/MIN y RCO, donde antes había una. Ambas se activan cuando el 74190 alcanza el último valor de su secuencia, ya sea 0 o 9, pero MAX/MIN se activa durante todo el ciclo del reloj, mientras que RCO sólo lo hace durante el nivel bajo del reloj (un semiciclo). En principio no parece útil RCO, pero sí lo es al conectar varios 74190. El cronograma de la figura 9-49 aclara la diferencia entre MAX/MIN y RCO.

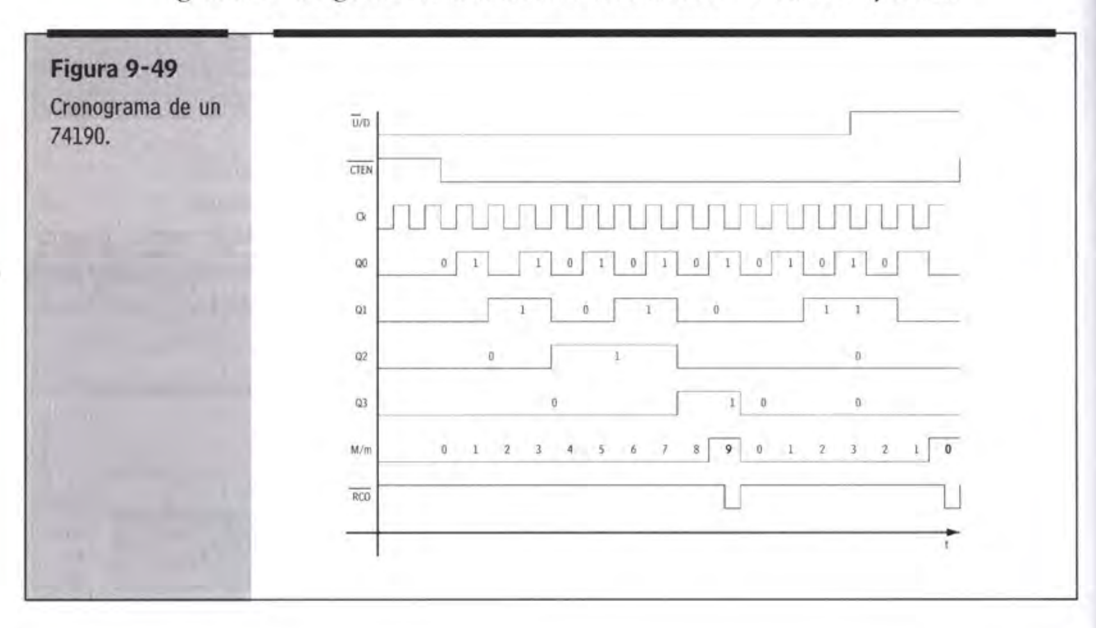


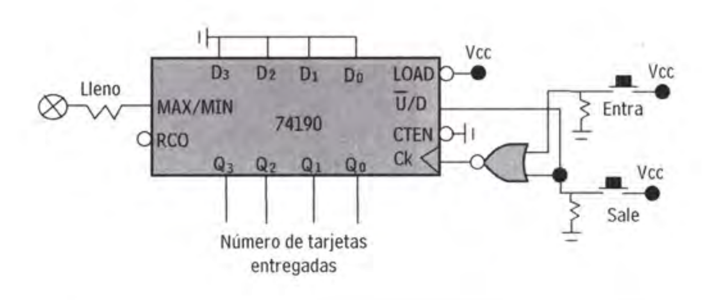
Figura 9-49
Cronograma de un 74190.

Ejemplo 9-3

Un vigilante a la puerta de un edificio dispone de 10 tarjetas de visita. Se desea diseñar el circuito que controle la máquina expendedora de tarjetas, sabiendo cuántas tarjetas ha dado y cuántas le han devuelto. Cada vez que un usuario solicita una tarjeta se genera un pulso, y lo mismo cuando devuelve la tarjeta.

El esquema de la figura 9-50 es una implementación del circuito de control.

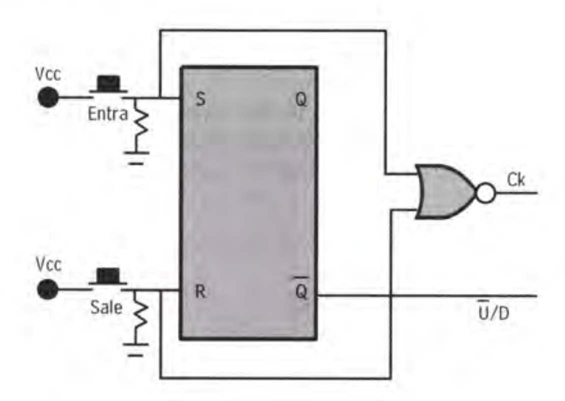
Figura 9-50
Circuito de control de visitas en un edificio.



Ejemplo 9-3 (Continuación)

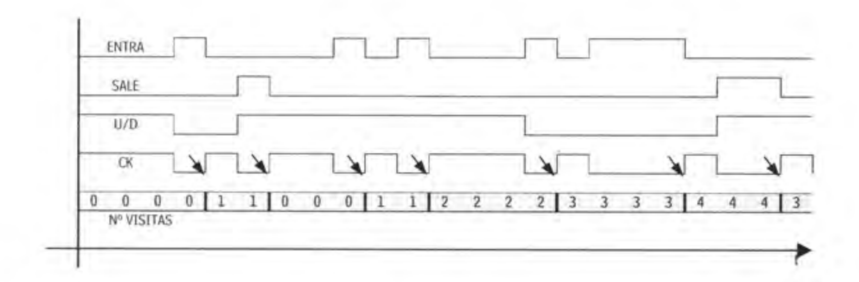
El hecho de que sea una puerta NOR y no OR consigue que el contador no cuente en el flanco de subida del pulsador, sino en el de bajada, de este modo cuando llegue el pulso ya se habrá estabilizado U/D. Aunque ahora puede pasar lo contrario, que cuando llegue el pulso a través de la NOR ya haya desaparecido el valor estable de U/D. La solución pasa por memorizar el valor de U/D como muestra la figura 9-51.

Figura 9-51
Circuito de memoria del control de visitas.



Ahora, el valor de U/D quedará enclavado hasta que no entre o salga una nueva persona del edificio, permitiendo el correcto funcionamiento del contador respecto del reloj obtenido en la NOR. El cronograma de la figura 9-52 describe una situación típica del contador.

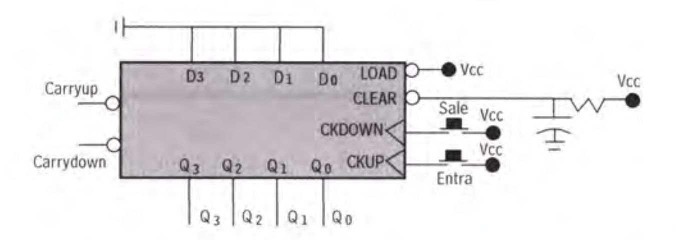
Figura 9-52
Cronograma del control de visitas en un edificio.



También podríamos haber utilizado el 74193, que dispone de dos relojes, uno de cuenta ascendente y otro descendente. En este caso, y a la vista de la figura circuito 9-53, el circuito se habría simplificado.

Ejemplo 9-3 (Continuación)

Figura 9-53
Circuito opcional con un 74193 del control de visitas.



El 74193 es un contador binario de módulo 16. Para conectar varios CI entre sí basta con unir el CARRYUP con el CKUP y el CARRYDOWN con el CKDOWN. Con este ejemplo se demuestra cómo a veces merece la pena dedicar algo de tiempo (o tener experiencia) a buscar el CI más adecuado a nuestras necesidades.

9.6.3. Otros contadores en CI

En los anteriores apartados hemos visto algunos de los contadores más comunes. En la tabla 9-12 aparecen listados y resumidos algunos contadores de la serie 74 con sus características más relevantes.

Tabla 9-12
Resumen de contadores en CI de la serie 74.

	Tipo	Sentido	Módulo	Clear	Preset	Carga	Enable	Tope
7490	A (a)	A (c)	10	SÍ	SÍ	NO	NO	NO
7492	A	A	12	SÍ	NO	NO	NO	NO
7493	A	A	16	SÍ	NO	NO	NO	NO
74160	S (b)	A	10	SÍ	NO	SÍ	SÍ	SÍ
74161	S	A	16	SÍ	NO	SÍ	SÍ	SÍ
74163	S	A	16	SÍ	NO	SÍ	SÍ	SÍ
74190	S	A/D	10	NO	NO	SÍ	SÍ	SÍ
74191	S	A/D	16	NO	NO	SÍ	SÍ	SÍ
74192	S	A/D	10	SÍ	NO	SÍ	NO	SÍ
74193	S	A/D	16	SÍ	NO	SÍ	NO	SÍ
74293	S	A	16	SÍ	NO	NO	NO	NO
74390	A	A	2x10 (d)	SÍ	NO	NO	NO	NO
74393	A	A	2x16	SÍ	NO	NO	NO	NO

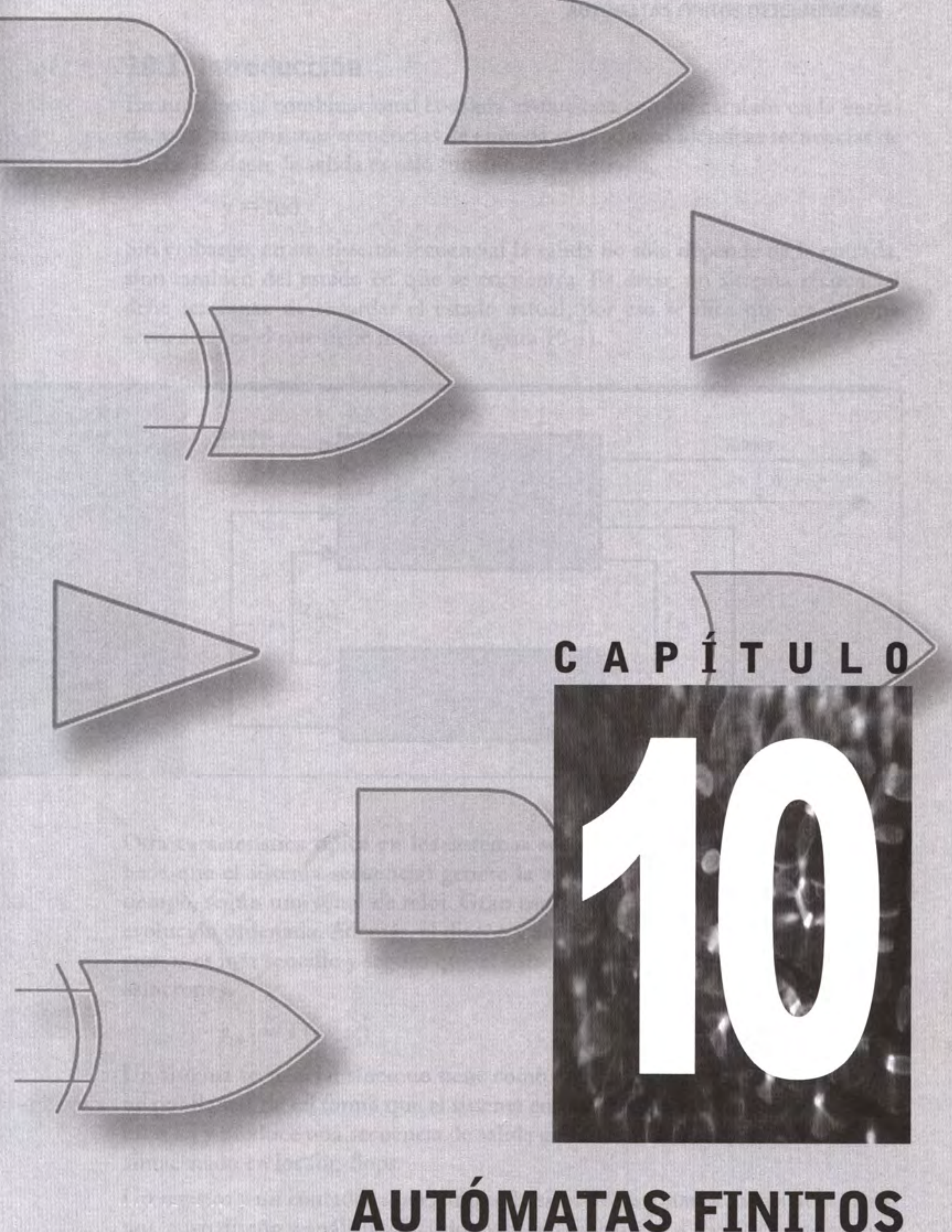
(a) Asíncrono, (b) Síncrono, (c) Ascendente y (d) Dos contadores

9.7. Resumen

Los contadores, junto con los registros, son los sistemas secuenciales más sencillos y utilizados. Los contadores se emplean para contar tiempo o eventos, para dividir frecuencias o para generar secuencias de control. Estas funciones pueden ser consideradas auxiliares en la mayoría de los sistemas, pero son muy comunes.

Desde un punto de vista teórico los contadores pueden implementarse siguiendo una estrategia síncrona o asíncrona. De estos dos caminos se obtienen resultados tanto prácticos como teóricos.

La utilidad y sencillez ya anotada de los contadores se refleja en la sencillez y variedad de los contadores MSI.



CAPÍTULO

10

AUTÓMATAS FINITOS DETERMINISTAS

10.1. Introducción

En un sistema combinacional la salida evoluciona con los cambios en la entrada, y así ante mismas secuencias de entrada se producen idénticas secuencias de salida. Es decir, la salida es sólo función de la entrada.

$$y = f(x)$$

Sin embargo, en un sistema secuencial la salida no sólo depende de la entrada, sino también del estado en que se encuentra. Es decir, un sistema secuencial debe ser capaz de recordar el estado actual, por eso se dice que un sistema secuencial es el que tiene memoria (figura 10-1).

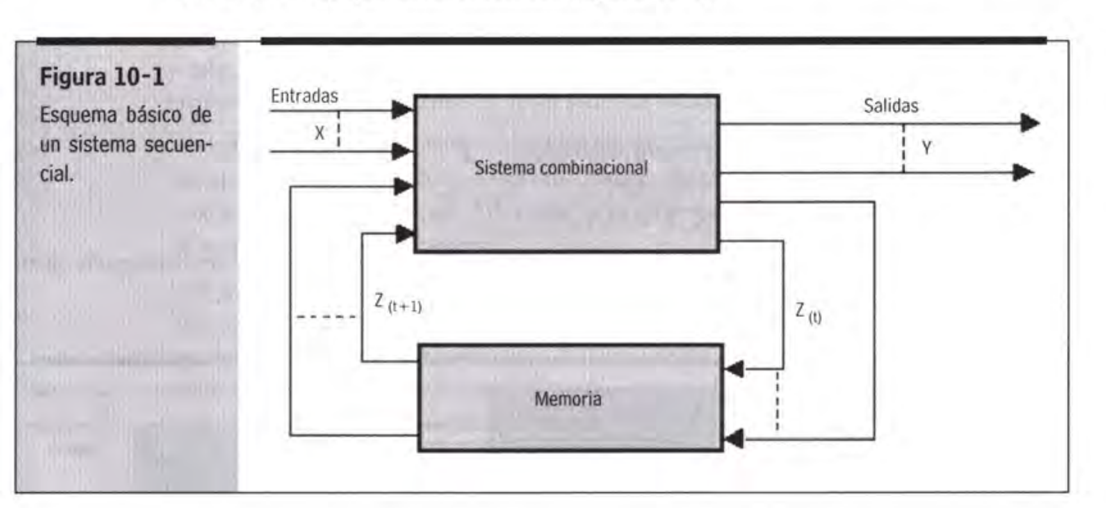


Figura 10-1
Esquema básico de un sistema secuencial.

Otra característica típica en los sistemas secuenciales es el sincronismo, que hace que el sistema secuencial genere la secuencia de forma ordenada en el tiempo, según una señal de reloj. Gran cantidad de sistemas necesitan de una evolución ordenada. Además, el diseño y análisis de circuitos secuenciales síncronos es más sencillo y seguro que el correspondiente a circuitos secuenciales asíncronos.

$$y_{t+1} = f(x_t, z_t)$$

Un sistema secuencial síncrono tiene como elemento básico al flip-flop síncrono por flanco, de tal forma que el sistema en su conjunto evoluciona al ritmo de un reloj y produce una secuencia de salida en función de la entrada y del estado almacenado en los flip-flops.

Un registro y un contador son ejemplos básicos de sistemas secuenciales síncronos, cuyo diseño y análisis ya ha sido visto en los capítulos 8 y 9. En este capítulo nos centraremos en el análisis y diseño de cualquier sistema secuencial síncrono, denominados también autómatas de estados finitos.

El desarrollo del capítulo será preferentemente metodológico, describiendo y sistematizando todos los pasos correspondientes a un análisis y un diseño correctos.

10.2. Autómatas de estados finitos. Modelos de Moore y Mealy

Un autómata puede ser contemplado según el modelo de Moore o de Mealy.

Si bien los autómatas de Mealy y Moore toman importancia cuando son utilizados para diseñar sistemas secuenciales síncronos, es necesario previamente definirlos de una forma más o menos estricta y conceptual, que nos permita aclarar las semejanzas y diferencias entre ambos modelos.

Autómata de estados finitos

Un autómata de estados finitos determinista o máquina secuencial (MS) se compone de:

X = Entradas
Y = Salidas
Z = Estados
δ = Transiciones entre estados
λ = Funciones de salida

$$MS = \langle X, Y, Z, \delta, \lambda \rangle$$

Los autómatas de Mealy (figura 10-2) y Moore (figura 10-3) se distinguen en su definición por cómo relacionan las entradas, salidas y estados.

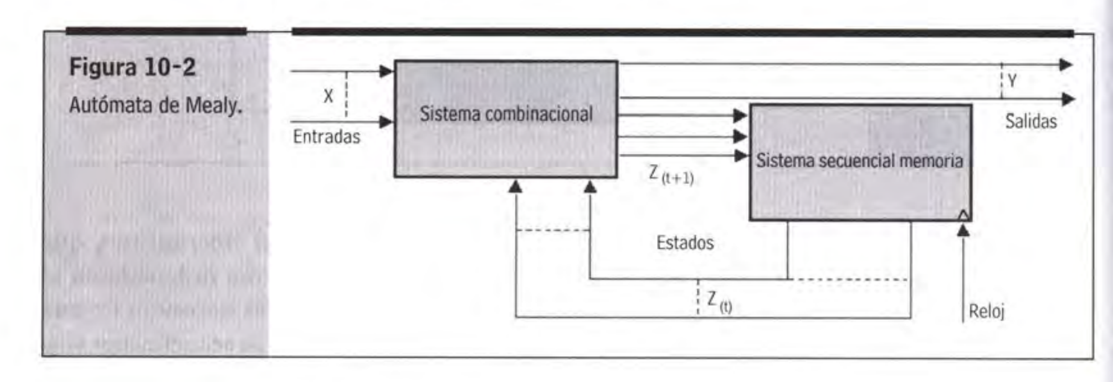


Figura 10-2
Autómata de Mealy.

Autómata de Mealy

$$Z = \delta\ (X, Z)$$
$$Y = \lambda\ (X, Z)$$

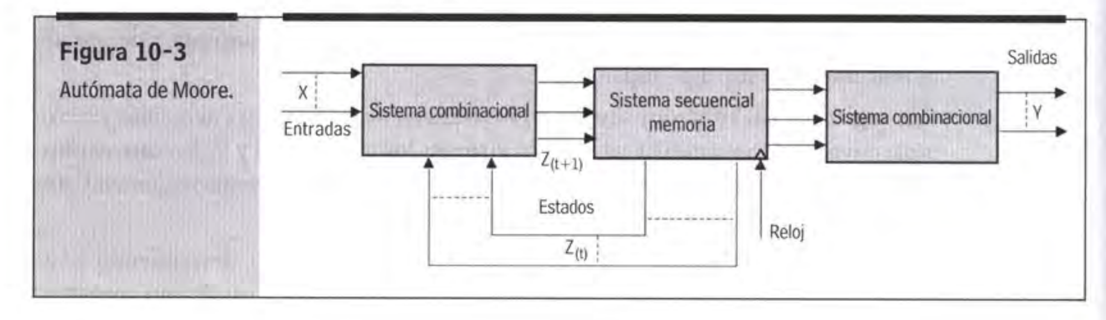


Figura 10-3
Autómata de Moore.

Autómata de Moore

$$Z = \delta\ (X, Z)$$
$$Y = \lambda\ (Z)$$

La diferencia está en cómo se obtiene la salida del autómata. En el modelo de Moore la salida sólo depende del estado, mientras que en el de Mealy depende de la entrada y del estado.

En un autómata de Moore la salida sólo evoluciona con el estado, pero como éste lo hace de forma síncrona, así la salida también evoluciona síncronamente; es decir, la salida sólo evolucionará en los correspondientes flancos de reloj. Sin embargo, en un autómata de Mealy, como la salida también depende la entrada y ésta no está sincronizada, resulta que la salida no variará exclusivamente síncronamente, sino al ritmo del reloj y de las variaciones de entrada.

Esta diferencia nos llevará a decidirnos en cada caso por un modelo Mealy o Moore; aunque muchas veces no existe diferencia entre lo implementado mediante Mealy o Moore. Además, todo autómata de Mealy se puede convertir en uno de Moore, y viceversa.

Los autómatas de Mealy y Moore se describen inicialmente mediante Diagramas de Transición de Estados o mediante Tablas de Transición de Estados (principalmente mediante diagramas). Un diagrama de transición de estados es un grafo que se define mediante Vértices y Arcos. Cada vértice es un estado y los arcos son transiciones entre estados. Las entradas se asocian a las transiciones y las salidas a los estados o a las transiciones, según sea un autómata de Moore o Mealy (figura 10-4), respectivamente.

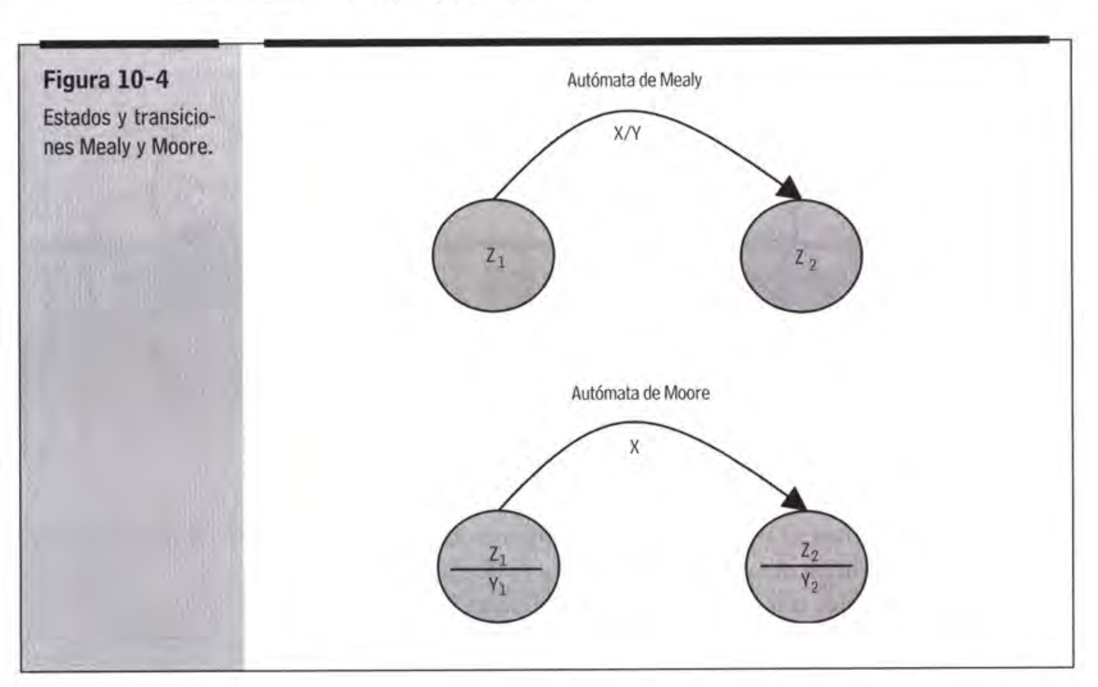


Figura 10-4
Estados y transiciones Mealy y Moore.

Veamos dos ejemplos de diagramas de transición de estados, el primero de Moore (figura 10-5) y el segundo de Mealy (figura 10-6).

Figura 10-5
Autómata de Moore.

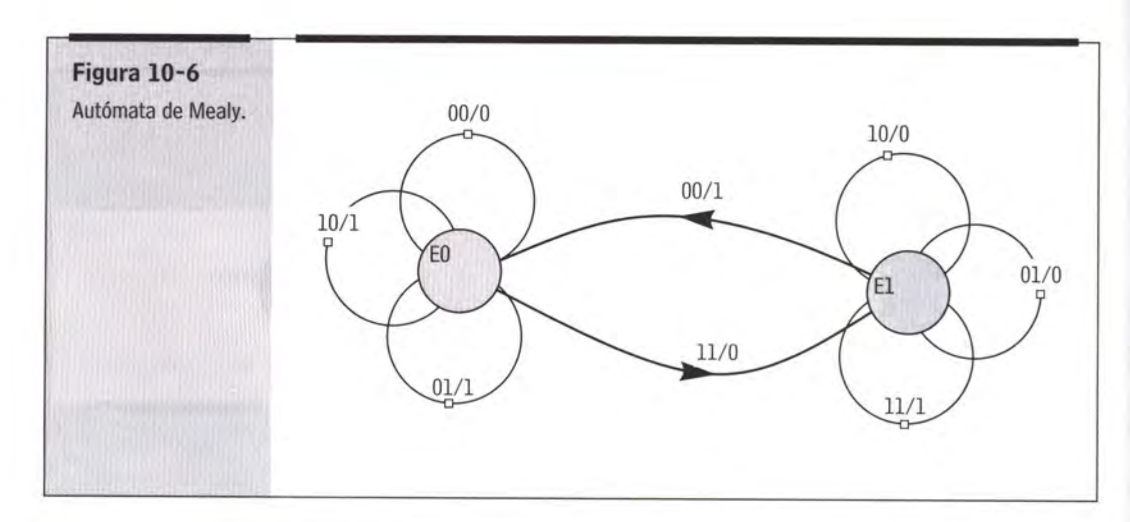


Figura 10-6
Autómata de Mealy.

Si leemos el diagrama correspondiente al autómata de Mealy, diremos que si estando en el estado E_0 la entrada es 00, la salida será 0 y el nuevo estado el E_0. También, que si estando en E_0 la entrada es 11 la salida será 0 (al igual que antes), pero el nuevo estado será E_1. Vemos también cómo las entradas 01 y 10

producen una salida 0 o 1, según esté en el estado E_1 o E_0, respectivamente, reafirmando el carácter secuencial del sistema.

En Moore vemos que si estamos en el estado E_0 la salida será 0, y si la entrada es 00, ni el estado ni la salida cambiarán. Si la entrada fuera 11, el nuevo estado sería E_2 y la salida 0. Y así sucesivamente.

Queda como trabajo para el lector comprobar que ambos autómatas se comportan de igual modo.

Ha quedado claro que describir textualmente el comportamiento de un sistema representado por su diagrama no resulta claro. La forma de sistematizar la información gráfica de un diagrama de transición de estados es mediante su tabla de transición de estados.

Dicha tabla tiene tantas filas como estados y tantas columnas como combinaciones haya de la entrada. Así, tendremos una tabla con tantas casillas como el producto del número de estados por el número de combinaciones de entrada. Completamos cada casilla con el nuevo estado que se corresponde a la transición desde E_j con la entrada X_i, es decir, Casilla i, j = $\delta (X_i, E_j)$. La salida irá en cada casilla o asociada a cada estado, según sea Mealy o Moore, respectivamente.

Visto desde la definición, obtener la tabla de transiciones es reordenar el diagrama en una tabla según la correspondencia:

$\delta: X \times E \rightarrow E$	$\delta: X \times E \rightarrow E$
$\lambda: X \times E \rightarrow Y$	$\lambda: E \rightarrow Y$
Mealy	Moore

Toda vez que X y E son conjuntos conocidos sólo queda plantear cada correspondencia en el diagrama propuesto. Por ejemplo $X = \{X_1, X_2, X_3, X_4\}$, $E = \{E_0, E_1\}$ y el autómata es de Mealy hay que plantear $\delta (X_1, E_0)$, $\delta (X_2, E_0)$ ... $\delta (X_4, E_0)$, $\delta (X_1, E_1)$ ... $\delta (X_4, E_1)$.

Obtengamos las tablas correspondientes a los dos ejercicios anteriores (tabla 10-1).

Tabla 10-1
Transiciones en Moore y Mealy.

MEALY	
$\delta (00, E_0) = E_0$	$\lambda (00, E_0) = 0$
$\delta (01, E_0) = E_0$	$\lambda (01, E_0) = 1$
$\delta (10, E_0) = E_0$	$\lambda (10, E_0) = 1$
$\delta (11, E_0) = E_1$	$\lambda (11, E_0) = 0$
$\delta (00, E_1) = E_0$	$\lambda (00, E_1) = 1$
$\delta (01, E_1) = E_1$	$\lambda (01, E_1) = 0$
$\delta (10, E_1) = E_1$	$\lambda (10, E_1) = 0$
$\delta (11, E_1) = E_1$	$\lambda (11, E_1) = 1$

Tabla 10-1 (Cont.)

Transiciones en Moore y Mealy.

MOORE	
$\delta(00, E_0) = E_0$	$\lambda(E_0) = 0$
$\delta(01, E_0) = E_1$	$\lambda(E_1) = 1$
$\delta(10, E_0) = E_1$	$\lambda(E_2) = 0$
$\delta(11, E_0) = E_2$	$\lambda(E_3) = 1$
$\delta(00, E_1) = E_0$	
$\delta(01, E_1) = E_1$	
$\delta(10, E_1) = E_1$	
$\delta(11, E_1) = E_2$	
$\delta(00, E_2) = E_1$	
$\delta(01, E_2) = E_2$	
$\delta(10, E_2) = E_2$	
$\delta(11, E_2) = E_3$	
$\delta(00, E_3) = E_1$	
$\delta(01, E_3) = E_2$	
$\delta(10, E_3) = E_2$	
$\delta(11, E_3) = E_3$	

Reordenemos lo anterior para obtener las correspondientes tablas de transición de estados y salida (tablas 10-2 y 10-3).

Autómata de Moore

Tabla 10-2

Tablas de un autómata de Moore.

		ENTRADAS			
		00	01	10	11
ESTADOS	E_0	E_0	E_1	E_1	E_2
	E_1	E_0	E_1	E_1	E_2
	E_2	E_1	E_2	E_2	E_3
	E_3	E_1	E_2	E_2	E_3

		SALIDA
		Y
ESTADOS	E_0	0
	E_1	1
	E_2	0
	E_3	1

Autómata de Mealy

Tabla 10-3
Tabla de un autómata de Mealy.

	ENTRADAS			
	00	01	10	11
E_0	$E_0/0$	$E_0/1$	$E_0/1$	$E_1/0$
E_1	$E_0/1$	$E_1/0$	$E_1/0$	$E_1/1$

Toda vez que hemos presentado qué es un autómata de estados finitos determinista, sus modelos de Mealy y Moore y su representación mediante diagramas y tablas, queda completar diferentes ejemplos que muestren los entresijos del diseño de sistemas secuenciales síncronos. Pero antes veamos cómo analizar autómatas.

10.3. Análisis de sistemas secuenciales

Antes de diseñar sistemas secuenciales síncronos es interesante plantear su análisis.

En este caso se parte de un circuito síncrono con memoria, es decir, con flip-flops, para obtener finalmente su diagrama de transición de estados o su correspondiente tabla de transición.

Analicemos, por ejemplo, el autómata de Mealy de la figura 10-7.

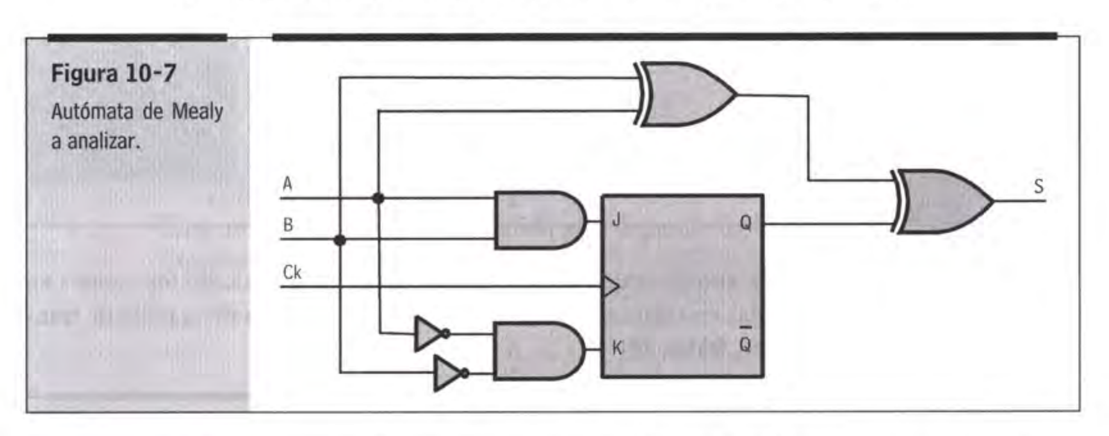


Figura 10-7
Autómata de Mealy a analizar.

Lo primero que observamos es que sólo hay un flip-flop y por tanto el autómata tiene dos estados: Z_0 y Z_1. Además, es un autómata de Mealy, pues la salida no sólo depende de Q, sino también de las entradas A y B. Resumiendo, conocemos los conjuntos X, Z e Y, pero no las relaciones δ y λ.

$$X = \{A, B\} = \{00, 01, 10, 11\},$$
$$Z = Q = \{0,1\} = \{Z_0, Z_1\},$$
$$Y = S = \{0,1\}$$

Las relaciones δ y λ conforman el diagrama de estados utilizando X, Z e Y. Para determinar δ y λ es necesario leer del circuito las ecuaciones de la parte combinacional de salida y de la parte combinacional de memoria. Así, resulta:

$$S = (A \oplus B) \oplus Q$$
$$J = A \cdot B$$
$$K = \overline{A} \cdot \overline{B}$$

Ahora que disponemos de las ecuaciones algebraicas debemos obtener su comportamiento para cada posible valor de A, B y Q, es decir, obtener algo parecido a una tabla de verdad que llamaremos tabla de comportamiento o excitación. Esta tabla tiene a la izquierda todos los posibles valores de Q, A y B en el instante t, a la derecha el valor de la salida S para cada combinación de Q, A y B, seguidamente el valor de J y K para cada combinación y por último el nuevo estado según el valor de Q_t y de J y K.

La obtención de la tabla 10-4 es tan laboriosa como sencilla.

Tabla 10-4
Tabla de excitación de Mealy.

	t					t + 1
Q	A	B	S	J	K	Q
0	0	0	0	0	1	0
0	0	1	1	0	0	0
0	1	0	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	0
1	0	1	0	0	0	1
1	1	0	1	0	0	1
1	1	1	1	1	0	1

Reescribimos la anterior tabla prescindiendo de J y K, y situando los estados en fila y las entradas en columna, es decir, según la disposición de la tabla de transición de estados (tabla 10-5).

Tabla 10-5
Tablas del autómata de Mealy analizado.

	A B			
Q	00	01	10	11
0	0/0	0/1	0/1	1/0
1	0/1	1/0	1/0	1/1

Q	Z
0	Z0
1	Z1

Opcionalmente podemos obtener el DTE de la figura 10-8. Para ello, primero damos un nombre a cada estado, por ejemplo Z_0 para Q = 0 y Z_1 para Q = 1

(ver la tabla de la derecha de 10-5), y luego sustituimos cada casilla por la correspondiente transición gráfica.

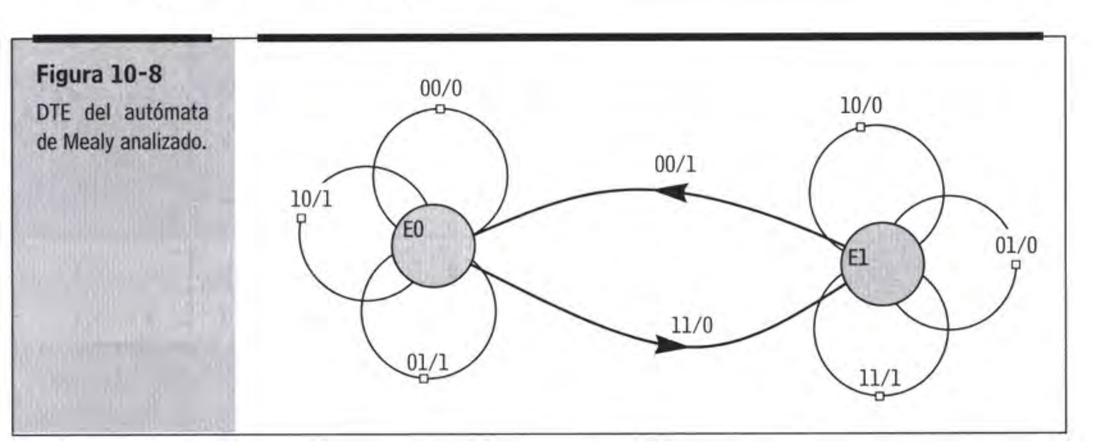


Figura 10-8
DTE del autómata de Mealy analizado.

Analicemos, por ejemplo, el autómata de Moore de la figura 10-9.

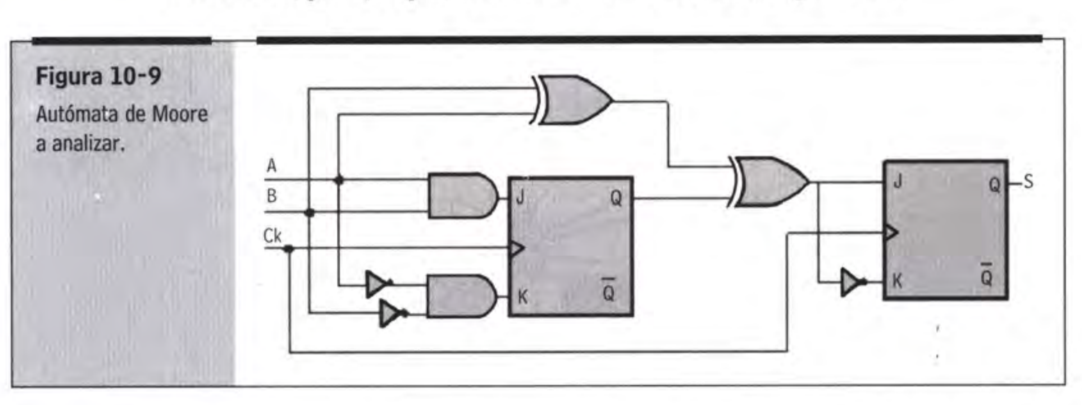


Figura 10-9
Autómata de Moore a analizar.

Este autómata de Moore -la salida sólo depende de Q_0- con dos flip-flops y dos entradas A y B. Así pues, tenemos:

$$X = \{A, B\} = \{00, 01, 10, 11\}$$
$$Z = \{Q_1, Q_0\} = \{00, 01, 10, 11\} = \{Z_0, Z_1, Z_2, Z_3\}$$
$$Y = S = \{0, 1\}$$

Las ecuaciones correspondientes a la salida y a las entradas J y K son:

$$S = Q_0$$
$$J_1 = A \cdot B \qquad J_0 = (A \oplus B) \oplus Q_1$$
$$K_1 = \overline{A} \cdot \overline{B} \qquad K_0 = \overline{(A \oplus B) \oplus Q_1}$$

La tabla de comportamiento tiene dos partes, puesto que es un autómata de Moore. La primera parte de la tabla 10-6 se refiere a los estados y la segunda a la salida.

Tabla 10-6

Tablas del autómata de Moore analizado.

t								t + 1	
Q_1	Q_0	A	B	J_1	K_1	J_0	K_0	Q_1	Q_0
0	0	0	0	0	1	0	1	0	0
0	0	0	1	0	0	1	0	0	1
0	0	1	0	0	0	1	0	0	1
0	0	1	1	1	0	0	1	1	0
0	1	0	0	0	1	0	1	0	0
0	1	0	1	0	0	1	0	0	1
0	1	1	0	0	0	1	0	0	1
0	1	1	1	1	0	0	1	1	0
1	0	0	0	0	1	1	0	0	1
1	0	0	1	0	0	0	1	1	0
1	0	1	0	0	0	0	1	1	0
1	0	1	1	1	0	1	0	1	1
1	1	0	0	0	1	1	0	0	1
1	1	0	1	0	0	0	1	1	0
1	1	1	0	0	0	0	1	1	0
1	1	1	1	1	0	1	0	1	1

Q_1	Q_0	S
0	0	0
0	1	1
1	0	0
1	1	1

Q_1	Q_0	Z
0	0	Z_0
0	1	Z_1
1	0	Z_2
1	1	Z_3

Reordenando la anterior tabla obtenemos la tabla 10-7 de transición de entradas y de salida.

Tabla 10-7

Tablas del autómata de Moore analizado.

	Entr.			
Est.	00	01	10	11
Z_0	Z_0	Z_1	Z_1	Z_2
Z_1	Z_0	Z_1	Z_1	Z_2
Z_2	Z_1	Z_2	Z_2	Z_3
Z_3	Z_1	Z_2	Z_2	Z_3

Est.	S
Z_0	0
Z_1	1
Z_2	0
Z_3	1

Tabla 10-7 (Cont.)

Tablas del autómata de Moore analizado.

		A B			
Q1	Q0	00	01	10	11
0	0	00	01	01	10
0	1	00	01	01	10
1	0	01	10	10	11
1	1	01	10	10	11

Q1	Q0	S
0	0	0
0	1	1
1	0	0
1	1	1

El diagrama resultante es el de la figura 10-10.

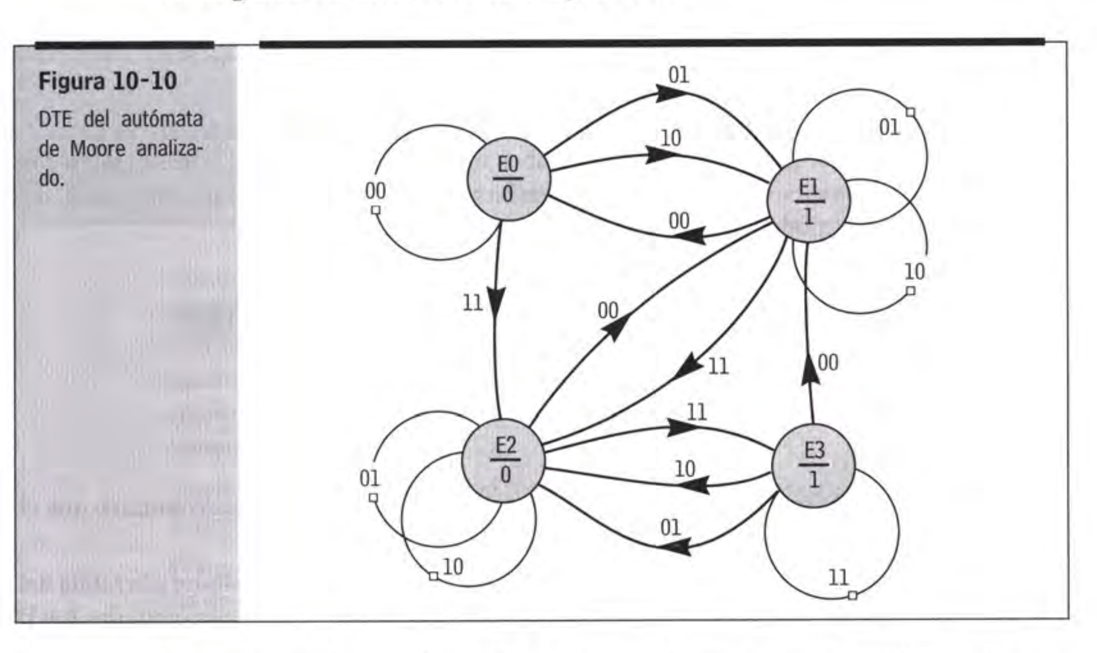


Figura 10-10

DTE del autómata de Moore analizado.

Como se puede ver, el anterior proceso es lento, aunque nos da una imagen completa del circuito objeto de análisis. Otra forma de analizar un circuito muy conocida es utilizando cualquier software de simulación.

Este software de simulación -por ejemplo el Electronic WorkBech- nos permite una cómoda captura del circuito y una más cómoda simulación de él, tanto a la hora de manejar el reloj y las entradas (interruptores, generador de palabras, etc.), como a la hora de observar las salidas (señalizadores, cronograma, etc.).

10.3.1. Comparación Moore vs Mealy

Ya hemos dicho que tanto el modelo de Mealy como el de Moore representan con exactitud una máquina de estados finitos. La principal diferencia entre ambos modelos está en su obtención de la salida. En el caso Mealy la salida depende del estado actual y de la entrada, así pues cualquier variación en la entrada o en el estado conlleva la variación de la salida. Sin embargo, en Moore

la salida sólo depende del estado actual, y como éste varía síncronamente con el reloj, la salida evolucionará también síncronamente.

En Moore, una combinación de entrada activa una transición para llegar a un nuevo estado, y con éste a una nueva salida. Sin embargo, en Mealy cada cambio en la entrada puede cambiar la salida, a la vez que prepara la transición para el nuevo flanco. Es decir, Moore es más ordenado y sistemático (entrada-transición-estado-salida), mientras que en Mealy la salida y los estados evolucionan por separado.

Como ya se ha dicho, toda máquina de estado se puede implementar mediante su autómata de Moore o de Mealy. La elección de uno u otro modelo depende de la propia funcionalidad del sistema y del gusto del diseñador, que en principio suele preferir el modelo de Mealy, ya que éste asegura diagramas con menos número de estados que el derivado del modelo de Moore.

Utilicemos el ejemplo anterior de las figuras 10-8 y 10-10 para aclarar las diferencias entre Moore y Mealy. Ambos diagramas de estado se corresponden con un sumador en serie que recibe los bits de A y B de uno en uno, entregando del mismo modo la salida.

Por ejemplo:

	t+5	t+4	t+3	t+2	t+1	t
A	0	1	0	1	1	0
B	0	1	1	1	0	0
S	1	1	0	0	1	0

Antes de seguir recordemos que el circuito de Mealy era más económico que el de Moore: dos estados en Mealy y cuatro en Moore.

Para observar las diferencias entre la salida del autómata de Moore y la salida del de Mealy planteamos el cronograma de la figura 10-11, donde las entradas A y B no están sincronizadas con el reloj.

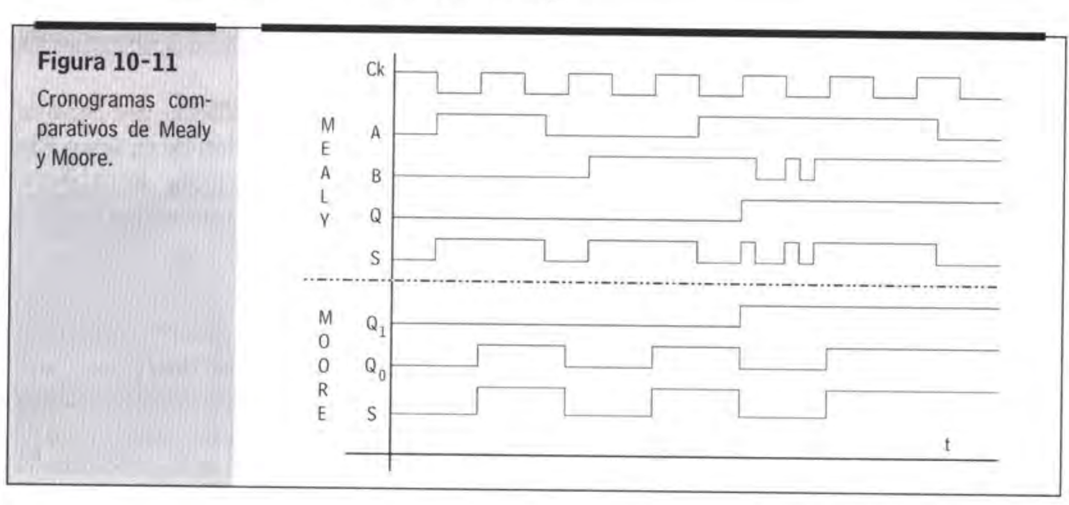


Figura 10-11

Cronogramas comparativos de Mealy y Moore.

En el caso de Mealy vemos cómo las variaciones de la entrada tienen efecto inmediato en la salida, mientras que la salida de Moore sólo contempla el valor de las entradas en el momento del flanco activo del reloj.

Veamos en el cronograma de la figura 10-12 que si las entradas están sincronizadas con el reloj, ambos autómatas ofrecen una misma salida. También vemos cómo un glitch en el último pulso afecta al autómata de Mealy, y no al de Moore.

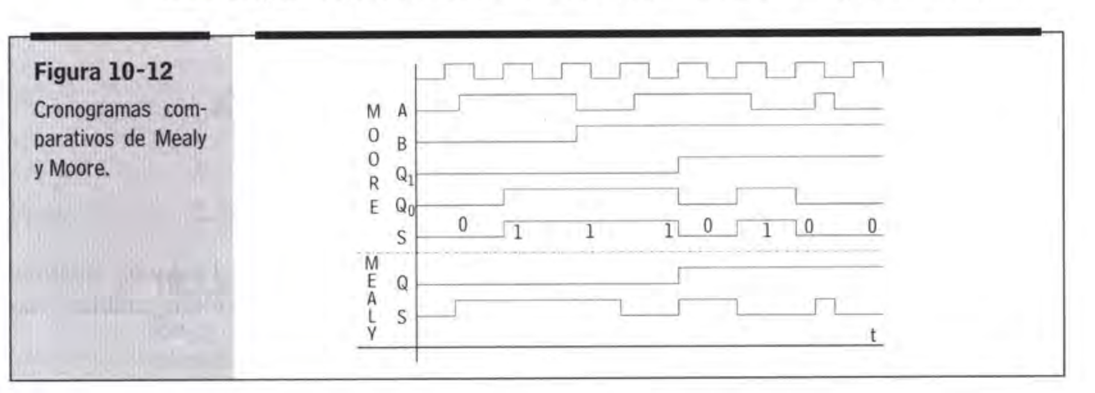


Figura 10-12
Cronogramas comparativos de Mealy y Moore.

Aunque las salidas de los cronogramas 10-11 y 10-12 presenten una misma secuencia, vemos cómo la S de Mealy se adelanta a la de Moore -o esta última se retrasa-; esto es porque Mealy asocia la salida a la transición -a lo que está pasando- y Moore asocia la salida al estado -a lo que ha pasado-. También vemos en este ejemplo cómo a Mealy le afecta el glitch del último pulso, mientras que a Moore no.

Queda como ejercicio para el lector completar un cronograma como el anterior donde el cambio en A y B se dé justo antes del flanco ascendente.

Veamos un ejemplo más. En las figuras 10-13 y 10-14 se presentan dos autómatas de Moore y Mealy cuya salida se pone a 1 al recibir, por su entrada serie X, 4 unos. Es decir, por cada cuatro unos en la entrada X, la salida se pone a 1 hasta que llega el siguiente 1 o 0.

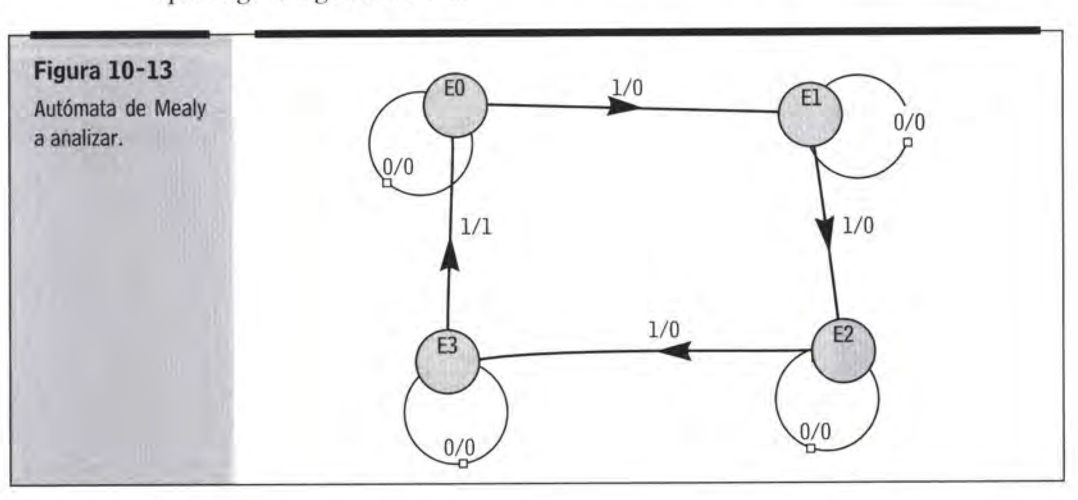


Figura 10-13
Autómata de Mealy a analizar.

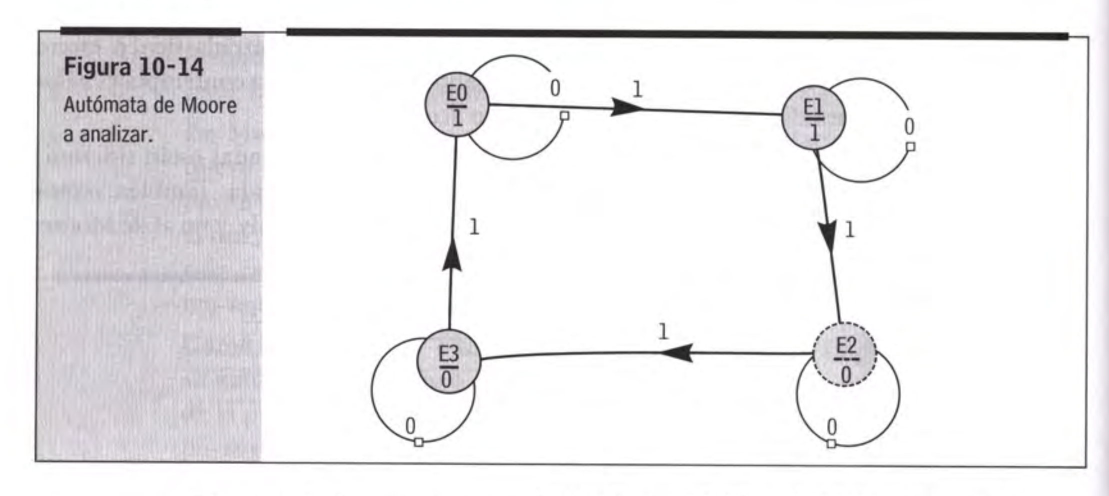


Figura 10-14
Autómata de Moore a analizar.

Observando los cronogramas de las figura 10-15 y 10-16 (el primero síncrono por flanco ascendente y el segundo por descendente) se podrán establecer las similitudes y diferencias entre ambas estrategias.

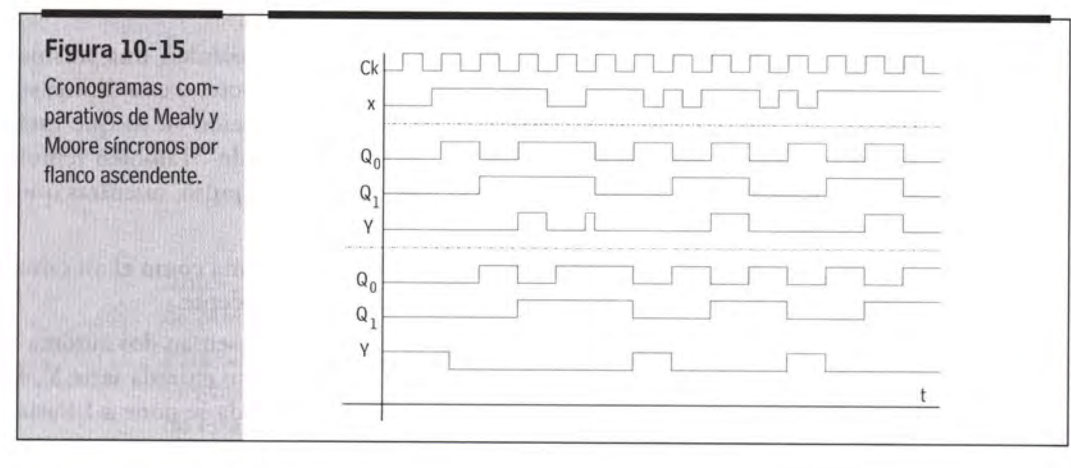


Figura 10-15
Cronogramas comparativos de Mealy y Moore síncronos por flanco ascendente.

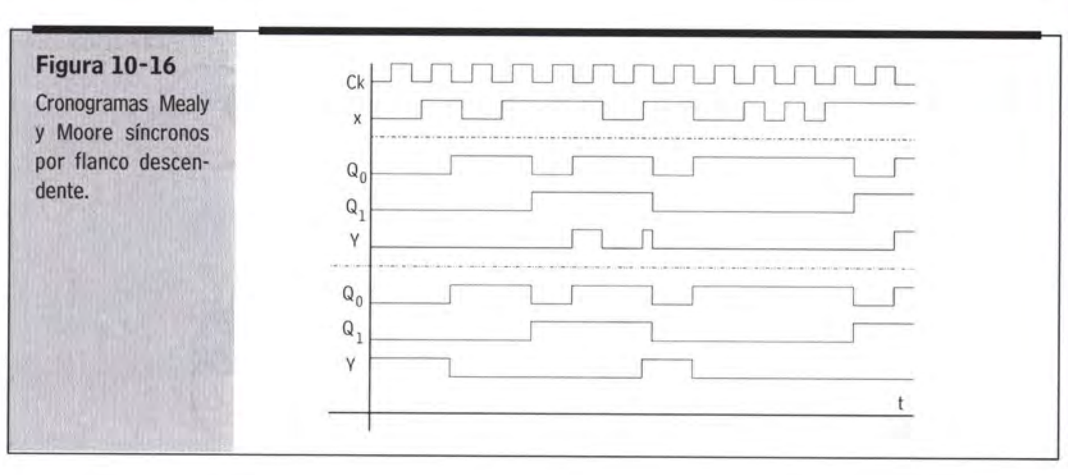


Figura 10-16
Cronogramas Mealy y Moore síncronos por flanco descendente.

Si tuviéramos que decidirnos por Moore o Mealy en este caso, y a la vista de los cronogramas, parece claro optar por Moore, ya que la salida sólo se pone a 1 cuando se han recibido cuatro unos desde el último 1 en la salida, no siendo éste el caso de Mealy (ver figura 10-16).

Resumamos las diferencias entre Moore y Mealy, recordando que ambos son modelos de la misma realidad y que siempre se puede pasar de Moore a Mealy, y viceversa. Así pues, la elección entre Moore y Mealy depende de la funcionalidad del sistema a diseñar y del gusto del diseñador. Recordemos estas diferencias:

- En Mealy la salida se asocia a la transición, en Moore al estado.
- La salida de Mealy está *adelantada* respecto a la de Moore.
- Un autómata de Mealy suele tener menos estados que el de Moore.
- Mealy es más sensible a glitches que Moore.

10.3.2. Transformación Mealy a Moore

En general, es un autómata de Mealy el que se transforma en uno de Moore. El paso de un autómata de Moore a uno de Mealy necesita de un rediseño completo.

Una vez que tenemos el diagrama de transición de estados de un autómata de Mealy, su transformación en uno de Moore pasa por una simple manipulación.

Debemos tener en cuenta que:

- Un estado de Mealy se transforma en tantos como transiciones con salidas distintas tenga.
- Asociamos a cada nuevo estado su salida, que es la correspondiente a las transiciones asociadas.
- Unir los nuevos estados entre sí respetando el diagrama original.
- Para cada nuevo estado se deben plantear todas las posibles transiciones, no sólo aquellas que tenían una misma salida.
- El resultado puede tener estados en exceso, que serán eliminados mediante el procedimiento de Reducción de Estados.

Veamos como ejemplo el caso del sumador serie de la figura 10-17.

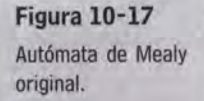
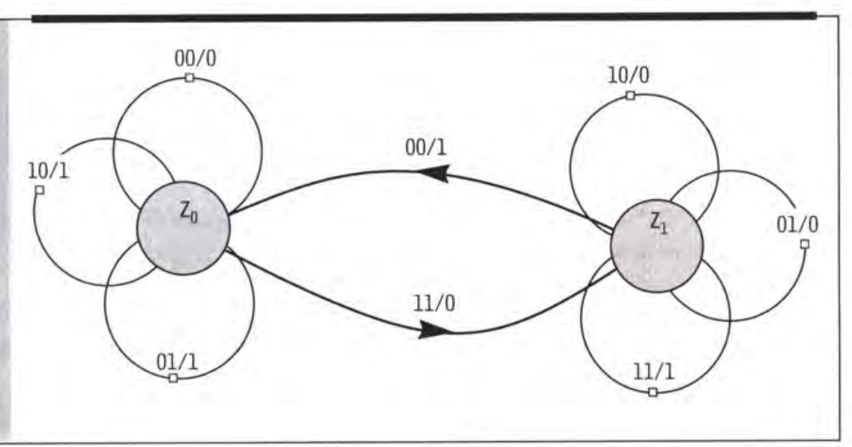


Figura 10-17
Autómata de Mealy original.



Vemos que tanto Z_0 como Z_1 tienen dos tipos de transiciones: unas con salida 0 y otras con salida 1 (figura 10-18), resultando los estados de la figura 10-19.

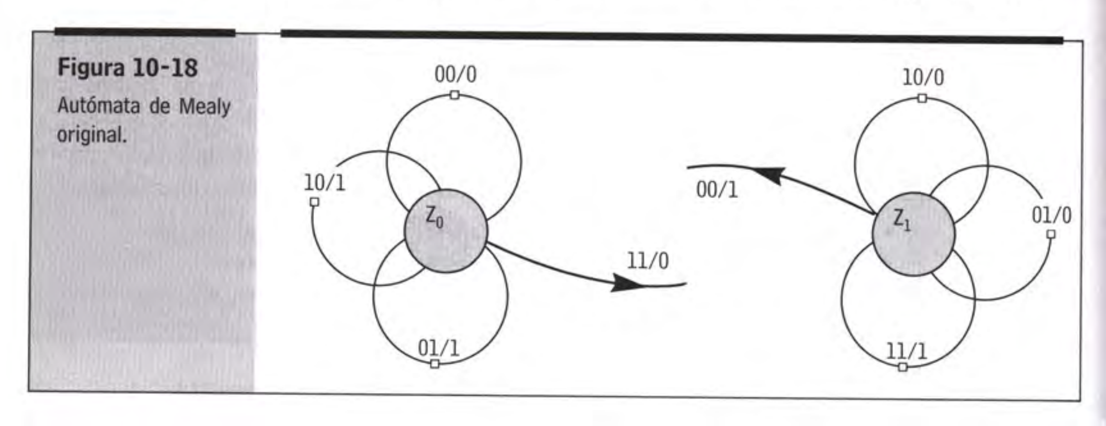


Figura 10-18
Autómata de Mealy original.

Figura 10-19
Estados del autómata de Moore.

Z_{0A} / 0

Z_{1A} / 1

Z_{0B} / 1

Z_{1B} / 0

Veamos cómo plantear las transiciones para Z_{0A} con S = 0 leyendo el diagrama original:

- Si AB = 00 entonces volvemos a Z_0 con S = 0, o sea, a Z_{0A}.
- Si AB = 01 entonces volvemos a Z_0 con S = 1, o sea, a Z_{0B}.
- Si AB = 10 entonces volvemos a Z_0 con S = 1, o sea, a Z_{0B}.
- Si AB = 11 entonces volvemos a Z_1 con S = 0, o sea, a Z_{1B}.

Así pues, el autómata se convierte en lo mostrado en la figura 10-20.

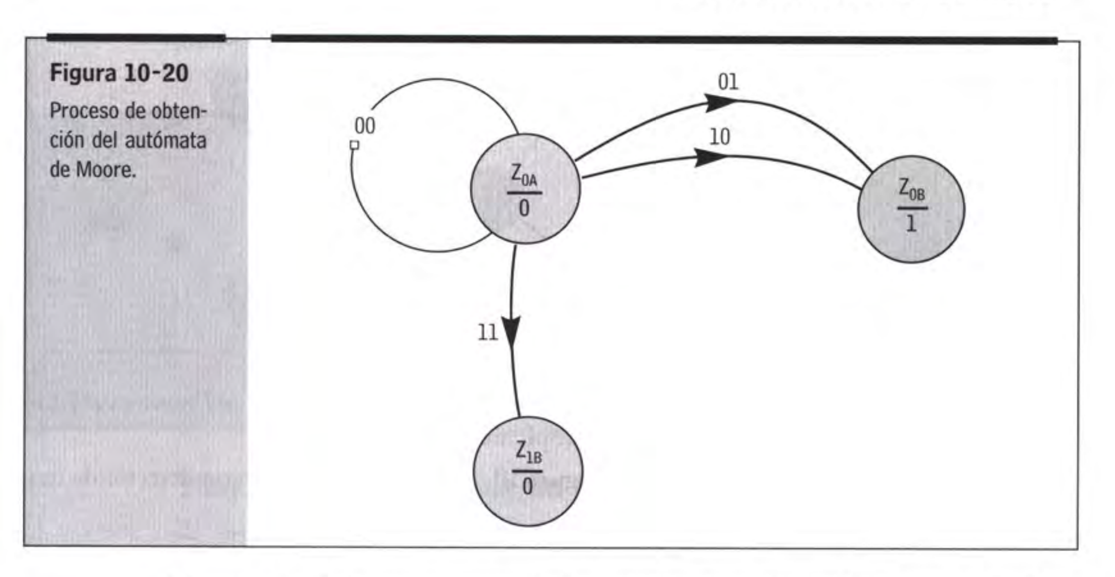


Figura 10-20
Proceso de obtención del autómata de Moore.

Queda al lector completar los estados restantes, y así comprobar que éste es idéntico al original planteado en la figura 10-10.

El ejemplo de la figura 10-21 es un autómata que siempre pasa de un estado al siguiente, sin entradas, es decir, cada estado sólo puede ir a otro. Al no haber entradas, falta esa parte en el DTE.

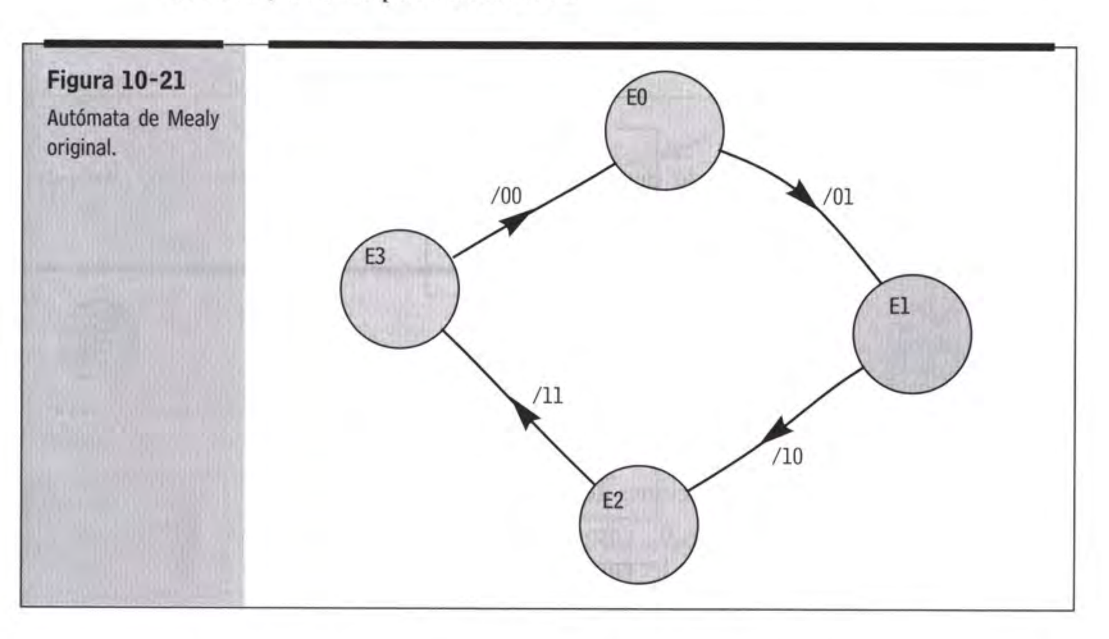


Figura 10-21
Autómata de Mealy original.

En este caso cada estado produce una salida única, por tanto cada estado se transforma en sí mismo y cada transición sigue siendo única. El resultado es la parte izquierda de la figura 10-22, que reordenado queda como lo mostrado en la derecha.

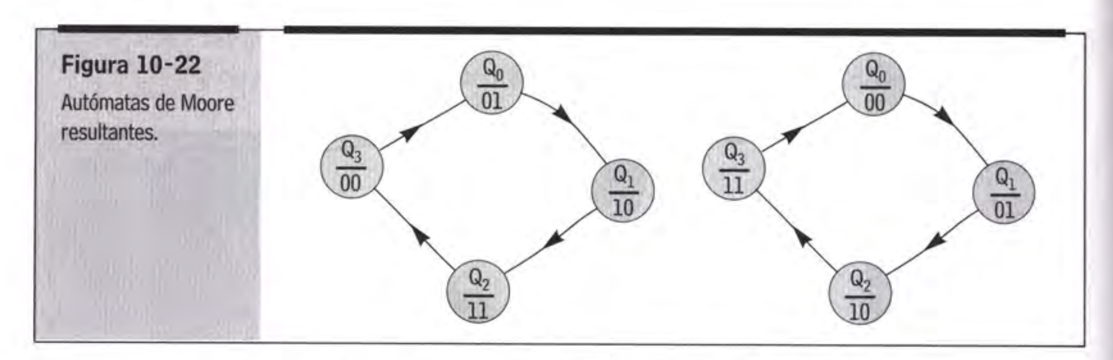


Figura 10-22
Autómatas de Moore resultantes.

En este caso el coste económico del autómata de Mealy y el de Moore es idéntico, de hecho son idénticas sus implementaciones.

Por último, el diagrama de la figura 10-23 se corresponde con un detector de tres o más unos consecutivos.

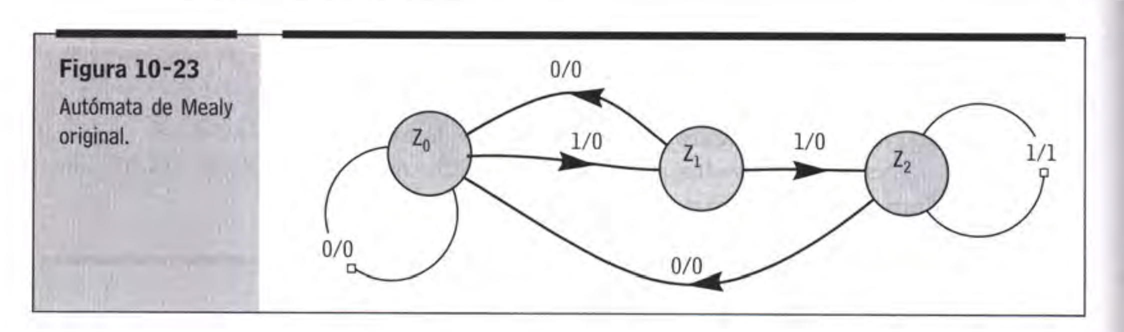


Figura 10-23
Autómata de Mealy original.

Sólo el último estado tiene dos salidas distintas en sus transiciones, así pues resulta la figura 10-24.

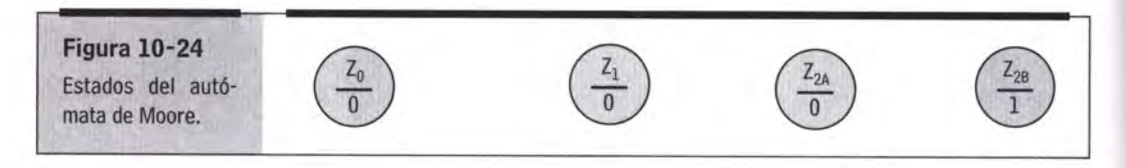


Figura 10-24
Estados del autómata de Moore.

Si tenemos las transiciones de Z_0, Z_1, Z_{2A} y Z_{2B}:

- Si en Z_0 $E = 0$ entonces el nuevo estado es Z_0 con $S = 0$
- Si en Z_0 $E = 1$ entonces el nuevo estado es Z_1 con $S = 0$
- Si en Z_1 $E = 0$ entonces el nuevo estado es Z_0 con $S = 0$
- Si en Z_1 $E = 1$ entonces el nuevo estado es Z_{2A} con $S = 0$
- Si en Z_{2A} $E = 0$ entonces el nuevo estado es Z_0 con $S = 0$
- Si en Z_{2A} $E = 1$ entonces el nuevo estado es Z_2 con $S = 1$, o sea Z_{2B}
- Si en Z_{2B} $E = 0$ entonces el nuevo estado es Z_0 con $S = 0$
- Si en Z_2B $E = 1$ entonces el nuevo estado es Z_2 con $S = 1$, o sea Z_{2B}

Completando el diagrama con las transiciones, el autómata de Moore resultante es el de la figura 10-25.

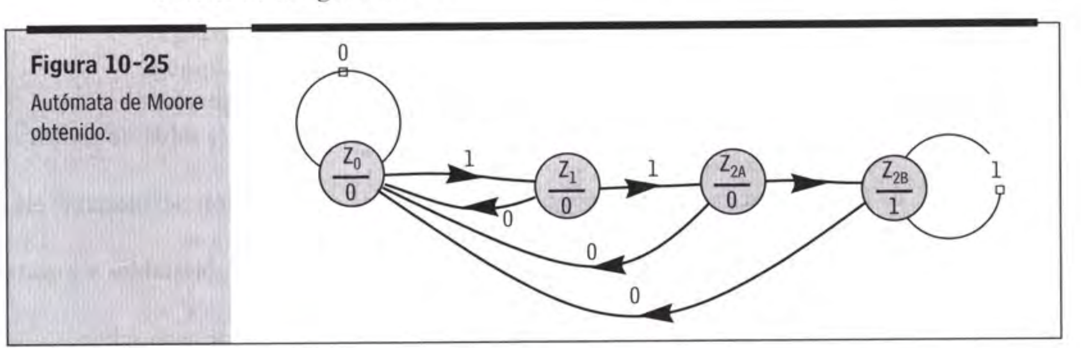


Figura 10-25
Autómata de Moore obtenido.

10.4. Diseño o síntesis de un autómata de estados finitos

El diseño de un autómata no es más que una transformación sistemática entre las distintas representaciones de un sistema secuencial síncrono (figura 10-26), de forma parecida a lo planteado para los sistemas combinacionales.

Figura 10-26
Diseño de un autómata.

Enunciado textual
Comprensión, experiencia e intuición
Diagrama de transición de estados
Reordenación
Tabla de transición de estados
Método de equivalencia entre estados
Tabla de transición de estados mínima
Codificación de estados
Tabla de transición de estados codificada
Tablas de verdad de los biestables
Tabla de excitación de los biestables
Diagramas de Veitch-Karnaugh
Expresiones simplificadas
Implementación
Circuito lógico y digital

En los sistemas secuenciales síncronos las representaciones pueden ser:

1. Descripción textual; que puede ser incompleta y/o incoherente.
2. Diagrama de transición de estados completo y coherente.
3. Tabla de transición de estados y de salida.
4. Tabla de transición de estados mínima (queda fuera del libro).
5. Tabla de transición de estados mínima codificada según la tabla de codificación de estados.
6. Tabla de excitación de biestables según la tabla de excitación del biestable elegido.
7. Diagrama de V-K correspondiente a cada entrada de los biestables y a cada salida.
8. Expresión simplificada de cada entrada de los biestables y de cada salida.
9. Circuito que implementa las anteriores expresiones.

El paso entre las distintas representaciones pasa por aplicar procedimientos sistemáticos que serán explicados en este capítulo. De la aplicación correcta de estos métodos se deriva un circuito correcto y mínimo.

Aunque de lo anterior parece claro que el diseñar autómatas es sencillo, existe un pero que lo complica todo: el paso del enunciado al diagrama de transición de estados.

Está claro que de la calidad del DTE dependerá la calidad final del circuito. Y la calidad del DTE depende de la experiencia e intuición del diseñador, además de la complejidad del propio sistema.

La experiencia se alcanza con ejemplos y con el propio diseño de autómatas. A continuación presentaremos varios ejemplos de diseño de autómatas que contemplan el diseño según las fases anteriores, excepto en dos. En los ejemplos no contemplaremos ni la reducción de estados para obtener la tabla mínima de transiciones ni plantearemos al completo el problema de la codificación.

Antes de comenzar los ejemplos quizá sean buenos algunos consejos a la hora de obtener un DTE del enunciado textual:

- Apoyarse en ejemplos sencillos, variados y completos para obtener el autómata al completo.
- Si en un estado con las mismas entradas se producen distintas situaciones y/o salidas, quiere decir que al menos son dos estados.
- En caso de duda añadir nuevos estados.
- Tener en cuenta los estados de restauración.

10.4.1. Diseño de autómatas de Mealy

Empecemos diseñando un sumador serie que recibe por las líneas A y B las entradas a sumar, sincronizadas con los flancos ascendentes del reloj.

1. Obtención del DTE

Veamos, un estado se distingue de otro por su comportamiento distinto ante entradas iguales. Por ejemplo, 0+0 no siempre da 0, ni 0+1 da siempre 1,

depende del acarreo o llevada recibido de la anterior suma. Así pues, podemos pensar que habrá dos estados: C_0 y C_1, llevada de valor 0 o 1, respectivamente.

A partir de lo anterior, veamos qué pasa en cada estado con cada entrada, por ejemplo:

- Si estando en C_0 la entrada es AB = 00 la salida será S = 0 y el nuevo acarreo será 0: C_0.
- Si estando en C_0 la entrada es AB = 10 la salida será S = 1 y el nuevo estado C_0, pues el acarreo sigue siendo cero.
- Si estando en C_0 la entrada es AB = 11 la salida será S = 0 y el nuevo estado C_1, pues el acarreo ahora es 1.
- Etc.

Resulta el DTE de la figura 10-27.

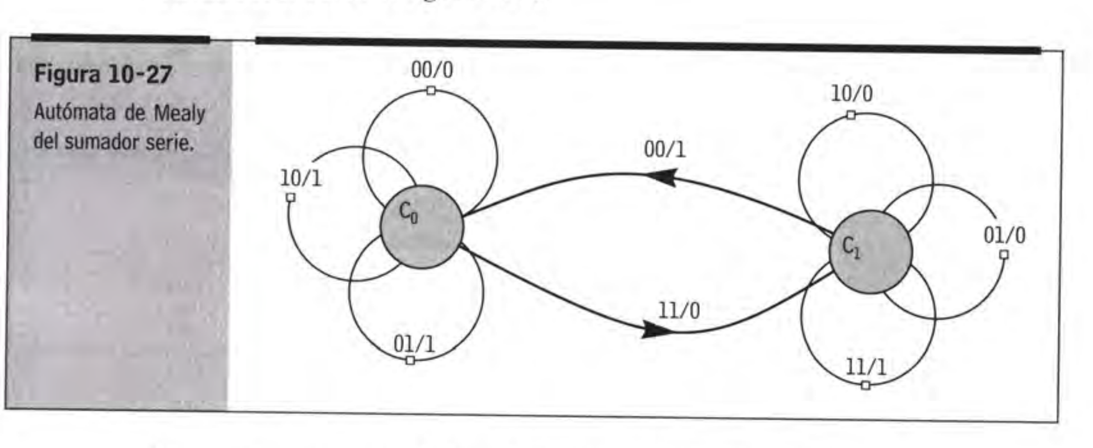


Figura 10-27
Autómata de Mealy del sumador serie.

Una vez que tenemos el DTE los restantes pasos no tienen problema.

2 y 3. Obtención de la Tabla de Transición de Estados. TTE

La TTE no es más que la reordenación en forma de tabla de la información gráfica del DTE. Es decir, es una transformación de gráfico a tabla, de hecho el DTE no es necesario, aunque sí es cómodo su uso para aclarar el diseño. La tabla 10-8 muestra a la TTE.

Tabla 10-8
Tabla de transición de estados.

	AB 00	01	10	11
C_0	$C_0/0$	$C_0/1$	$C_0/1$	$C_1/0$
C_1	$C_0/1$	$C_1/0$	$C_1/0$	$C_1/1$

Cada casilla se completa planteando a la frase: si estando en el estado Z_i la entrada es X_i, ¿cuáles serán el nuevo estado y salida? Por ejemplo, para la primera casilla: estando en C_0 si la entrada es 00 el nuevo estado será C_0 y la salida será 0. Y así sucesivamente para cada casilla.

En estos ejemplos prescindiremos de la minimización de estados, por otra parte imposible en este ejemplo.

4 y 5. Tabla de Transición de Estados Codificados

En este paso debemos codificar en binario cada estado, asignándole a cada uno una combinación de bits, llamados variables de estado Q_i. En la tabla 10-9 asignamos a C_0 el 0 y a C_1 el 1, aunque podríamos haberlo hecho al revés.

Tabla 10-9
Tabla de codificación de estados.

	Q
C_0	0
C_1	1

Seguidamente, sustituimos cada estado por su codificación para obtener la tabla 10-10 de transición de estados codificada.

Tabla 10-10
Tabla de transición de estados codificada.

	AB			
Q	00	01	10	11
0	0/0	0/1	0/1	1/0
1	0/1	1/0	1/1	1/1

6. Tabla de Excitación de Biestables

Previamente recordemos las tablas de excitación de los Biestables RS, J-K, D y T. En ellas planteamos lo siguiente: si Q_t es 0 y Q_{t+1} debe ser 1, ¿qué valores deberán tener J y K o R-S, D? Las tablas aparecen en 10-11 y deberán ser tenidas en cuenta todo el capítulo.

Tabla 10-11
Tablas de excitación de biestables.

Q_t	Q_{t+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Q_t	Q_{t+1}	S	R
0	0	0	x
0	1	0	1
1	0	1	0
1	1	x	0

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

En la Tabla de Excitación primero reordenamos en filas el panel de TTE y finalmente asociamos los valores de las entradas de los biestables. En la tabla 10-12 usaremos biestables J-K.

Tabla 10-12
Tabla de excitación del autómata.

t			t+1			
Q	A	B	S	Q	J	K
0	0	0	0	0	0	x
0	0	1	1	0	0	x
0	1	0	1	0	0	x
0	1	1	0	1	1	x
1	0	0	1	0	x	1
1	0	1	0	1	x	0
1	1	0	0	1	x	0
1	1	1	1	1	x	0

Primero obtendremos las columnas S y Q. La primera fila dice: siendo el estado 0 y la entrada 00, el nuevo estado es el 0 y la salida 1. Así para todas las filas.

Después se completan las columnas J y K atendiendo a Q_t y Q_{t+1}. En la primera fila decimos que si Q_t es 0 y Q_{t+1} debe ser 0, entonces J y K deben ser 0 y X (ver la tabla de excitación en 10-11).

7 y 8. Obtención de las expresiones mínimas

Tenemos la tabla de excitación anterior donde están descritas las J y K necesarias para el biestable y la salida. Así pues, podemos obtener sus correspondientes diagramas de V-K y simplificarlos (figura 10-28).

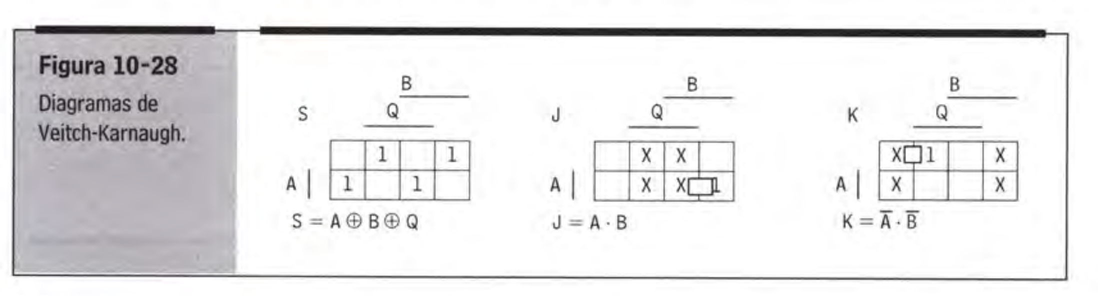


Figura 10-28
Diagramas de Veitch-Karnaugh.

9. Implementación del circuito y análisis

Queda finalmente implementar el circuito de la figura 10-29 y analizarlo mediante un software de simulación a fin de comprobar su buen funcionamiento.

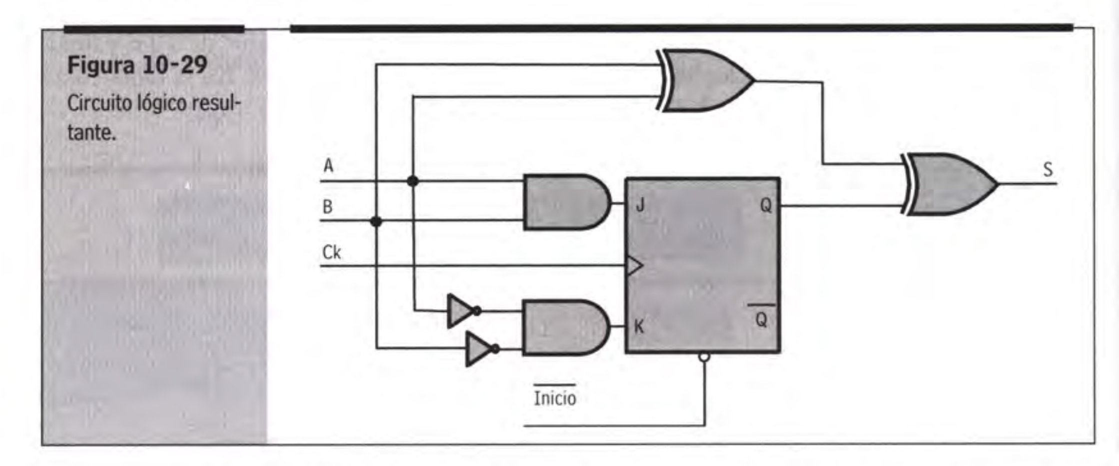


Figura 10-29
Circuito lógico resultante.

La línea de INICIO se encarga de restaurar el sumador para comenzar una nueva suma. La restauración debe dejar el acarreo en 0, lo que se consigue activando la línea Clear del J-K.

Pasemos a otro ejemplo, diseñemos el autómata de Mealy que activa la salida S si en la entrada A se han recibido tres o más unos consecutivamente.

1. Obtención del DTE

Vemos en el DTE de la figura 10-30 que han de llegar tres unos para que la salida se ponga a 1. También queda claro que un 0 hace que se *pierda la secuencia.*

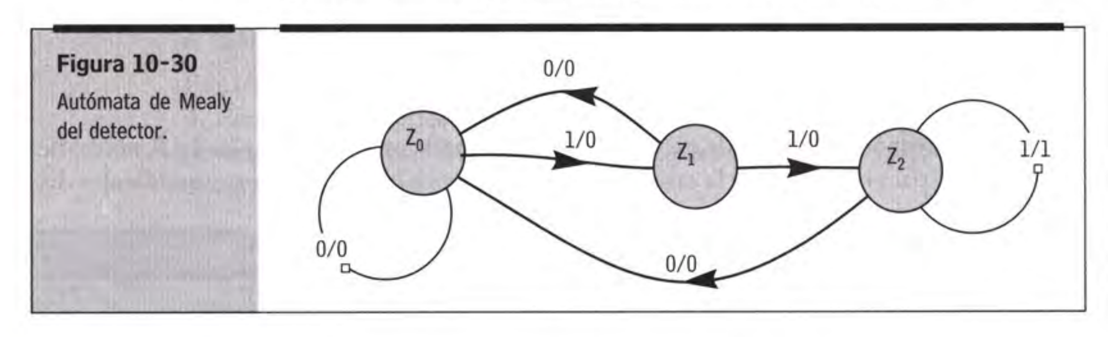


Figura 10-30
Autómata de Mealy del detector.

2. Obtención de la Tabla de Transición de Estados

La tabla 10-13 muestra la tabla de transición de estados del DTE.

Tabla 10-13
Tabla de transición de estados.

	A: 0	1
Z_0	$Z_0/0$	$Z_1/0$
Z_1	$Z_0/0$	$Z_2/0$
Z_2	$Z_0/0$	$Z_2/1$

3. Tabla de Transición de Estados Codificada

En este caso, la codificación se intuye más variada y compleja. Para tres estados necesitaremos dos variables de estado con sus cuatro combinaciones. El problema reside en qué combinación se debe asignar a cada estado; este problema es el de Asignación de Estados. En nuestro caso optamos por la codificación intuitiva y sencilla de la tabla 10-14.

Tabla 10-14
Tabla de transición y de codificación de estados.

A	Q_1	Q_0
Z_0	0	0
Z_1	0	1
Z_2	1	0
X	1	1

Q	Q_0	A: 0	A: 1
0	0	00/0	01/0
0	1	00/0	10/0
1	0	00/0	10/1
1	1	xx/x	xx/x

Como el estado de 11 no se puede dar, le asignamos condiciones libres. También podríamos haber optado por suponer que si se da 11 es a causa de un error en el autómata, y así forzaríamos el nuevo estado a 00, por ejemplo.

4. Tabla de excitación de los biestables

En la tabla 10-15 reescribimos la 10-14 utilizando biestables tipo D y recordando la tabla 10-11.

Tabla 10-15
Tabla de excitación del autómata.

t				t+1			
Q_1	Q_0	A	S	Q_1	Q_0	D_1	D_0
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	1
0	1	0	0	0	0	0	0
0	1	1	0	1	0	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	0
1	1	0	x	x	x	x	x
1	1	1	x	x	x	x	x

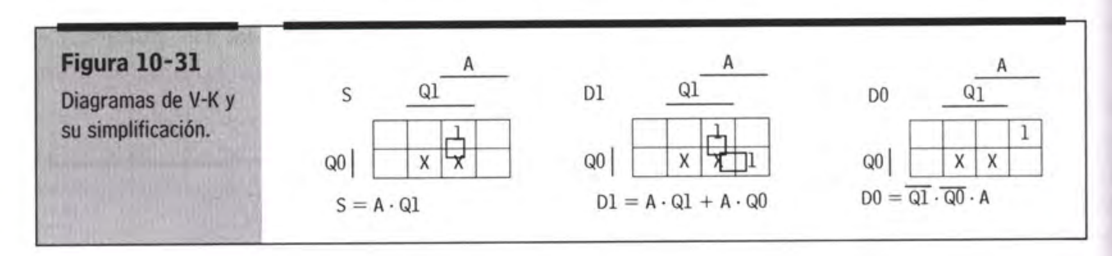
Como vemos en las filas 110 a 111, de 10-15, hemos optado por considerar imposible el estado 11, y asignarle condiciones libres.

5. Obtención de las expresiones mínimas

Dibujemos y simplifiquemos los diagramas de V-K de la figura 10-31.

Figura 10-31

Diagramas de V-K y su simplificación.

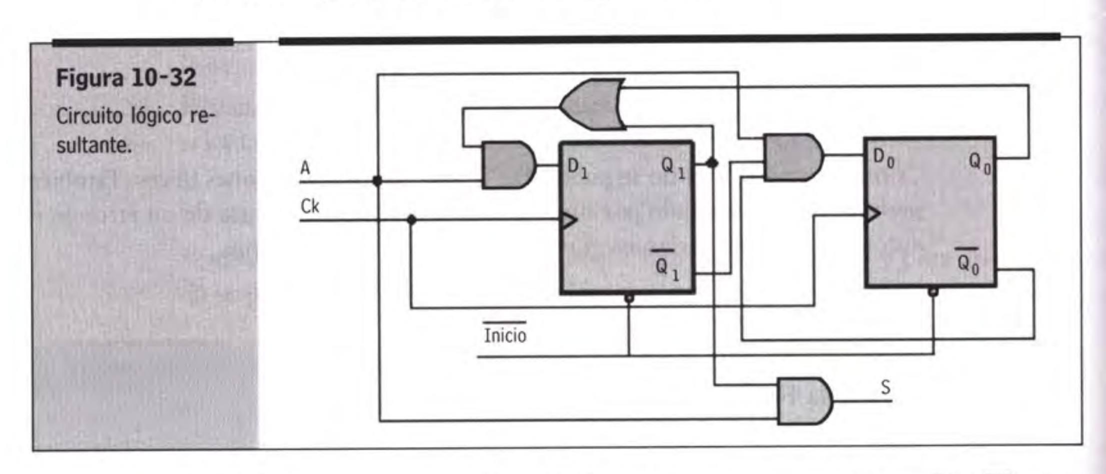


6. Implementación del circuito y análisis

El circuito queda como muestra la figura 10-32.

Figura 10-32

Circuito lógico resultante.



$S = Q_1 \cdot A$ $\qquad D_1 = A\,(Q_1 + Q_0)$ $\qquad D_0 = \overline{Q_1} \cdot \overline{Q_0} \cdot A$

Repitamos los pasos 4, 5 y 6 para los biestables J-K (tabla 10-16 y figura 10-33).

Tabla 10-16

Tabla de excitación del autómata.

	t			t+1					
Q_1	Q_0	A	S	Q_1	Q_0	J_1	K_1	J_0	K_0
0	0	0	0	0	0	0	X	0	0
0	0	1	0	0	1	0	X	1	1
0	1	0	0	0	0	0	X	X	0
0	1	1	0	1	0	1	X	X	0
1	0	0	0	0	0	X	1	0	0
1	0	1	1	1	0	X	0	0	0
1	1	0	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X

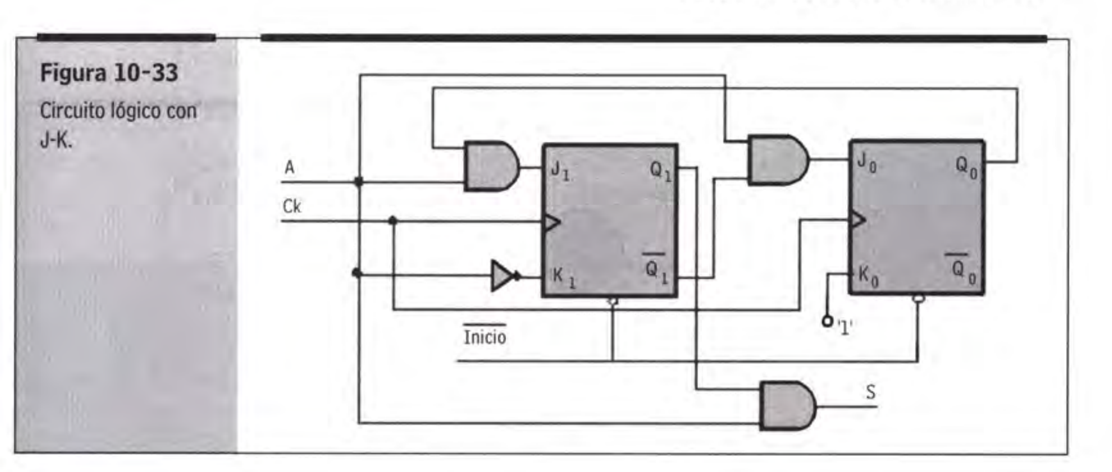
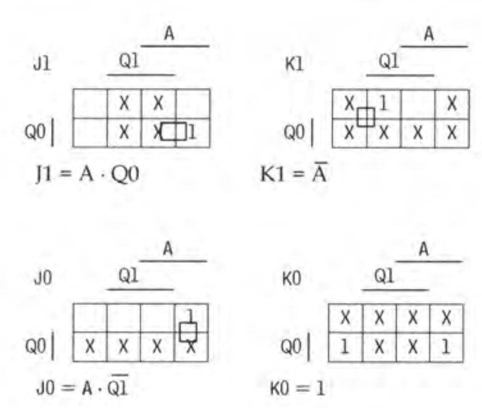


Figura 10-33
Circuito lógico con J-K.



Vemos que aunque este diseño es más laborioso en su proceso, resulta un circuito de menor coste que el correspondiente a biestables D.

10.4.2. Diseño de autómatas de Moore

Básicamente, el proceso es idéntico al planteado para autómatas de Mealy, excepto en el diseño de la parte correspondiente a la salida. La salida no formará parte de la tabla de transición de estados y demás, sino que formará una tabla propia, llamada tabla de salida.

Aplicando el método enunciado, diseñemos el autómata de Moore que implementa un sumador serie de dos líneas de datos A y B sincronizados con la línea del reloj.

1. Obtención del DTE

El DTE tiene que tener al menos tantos estados como salidas distintas. Pero también debe estudiar si aun siendo iguales las salidas pueden desdoblarse en varios estados. Éste es el caso del sumador serie, ya que es distinto que una suma dé 0 por ser 0+0+0 a que dé 0 por ser 1+1+0, ya que en este último caso habrá acarreo y en el anterior no.

El DTE queda como refleja la figura 10-34.

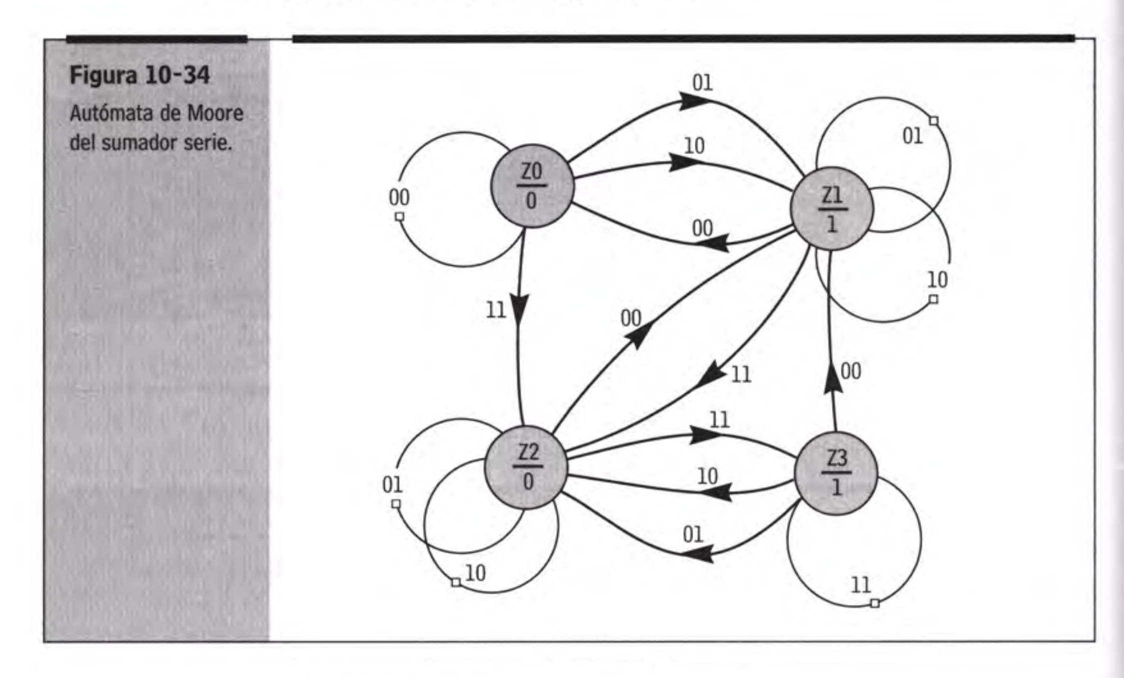


Figura 10-34
Autómata de Moore del sumador serie.

2. Tabla de Transición de Estados y Tabla de Salida

La tabla 10-17 muestra la tabla de transición de estados del DTE.

Tabla 10-17
Tabla de transición de estados y salida.

	AB			
	00	01	10	11
Z_0	Z_0	Z_1	Z_1	Z_2
Z_1	Z_0	Z_1	Z_1	Z_2
Z_2	Z_1	Z_2	Z_2	Z_3
Z_3	Z_1	Z_2	Z_2	Z_3

Est.	S
Z_0	0
Z_1	1
Z_2	0
Z_3	1

3. Tabla de Transición de Estados y Salida Codificados

Este caso es sencillo; en la tabla 10-18 asignamos a priori a cada estado Z_0-Z_3 el par de bits de su subíndice.

Tabla 10-18
Tabla de codificación y de transición codificada.

Estado	Q_1	Q_0	S
Z_0	0	0	0
Z_1	0	1	1
Z_2	1	0	0
Z_3	1	1	1

			A	B		
Q_1	Q_0	00	01	10	11	S
0	0	00	01	01	10	0
0	1	00	01	01	10	1
1	0	01	10	10	11	0
1	1	01	10	10	11	1

Hemos aprovechado la tabla de codificación para unirle la columna de salida.

4. Tabla de excitación de biestables

Ahora hay que obtener la tabla 10-19 de excitación para biestables J-K como se ha hecho para Mealy. La tabla correspondiente a la salida queda como estaba.

Tabla 10-19
Tabla de excitación del autómata.

Q_1	Q_0	A	B	Q_1	Q_0	J_1	K_1	J_0	K_0
0	0	0	0	0	0	0	X	0	X
0	0	0	1	0	1	0	X	1	X
0	0	1	0	0	1	0	X	1	X
0	0	1	1	1	0	1	X	0	X
0	1	0	0	0	0	0	X	X	1
0	1	0	1	0	1	0	X	X	0
0	1	1	0	0	1	0	X	X	0
0	1	1	1	1	0	1	X	X	1
1	0	0	0	0	1	X	1	1	X
1	0	0	1	1	0	X	0	0	X
1	0	1	0	1	0	X	0	0	X
1	0	1	1	1	1	X	0	1	X
1	1	0	0	0	1	X	1	X	0
1	1	0	1	1	0	X	0	X	1
1	1	1	0	1	0	X	0	X	1
1	1	1	1	1	1	X	0	X	0

5. Simplificación de funciones

Los diagramas de V-K de la figura 10-35 de las entradas J-K tienen cuatro variables: dos de estado y dos de salida. Sin embargo, el V-K de salida sólo tiene dos variables, las correspondientes al estado.

Figura 10-35
Diagramas V-K simplificados.

J1 (A, Q1 / Q0, B)

	X	X	
	X	X	
	X	X	1
	X	X	1

J1 = A · B

K1 (A, Q1 / Q0, B)

X	1		X
X	1		X
X			X
X			X

$K1 = \overline{A} \cdot \overline{B}$

J0 (A, Q1 / Q0, B)

	1		1
X	X	X	X
X	X	X	X
1		1	

$J0 = A \oplus B \oplus Q1$

K0 (A, Q1 / Q0, B)

X	X	X	X
1		1	
	1		1
X	X	X	X

$K0 = \overline{A \oplus B \oplus Q1}$

S (Q1 / Q0)

1	1

S = Q0

6. Implementación del Circuito y Análisis

El circuito resultante es el de la figura 10-36.

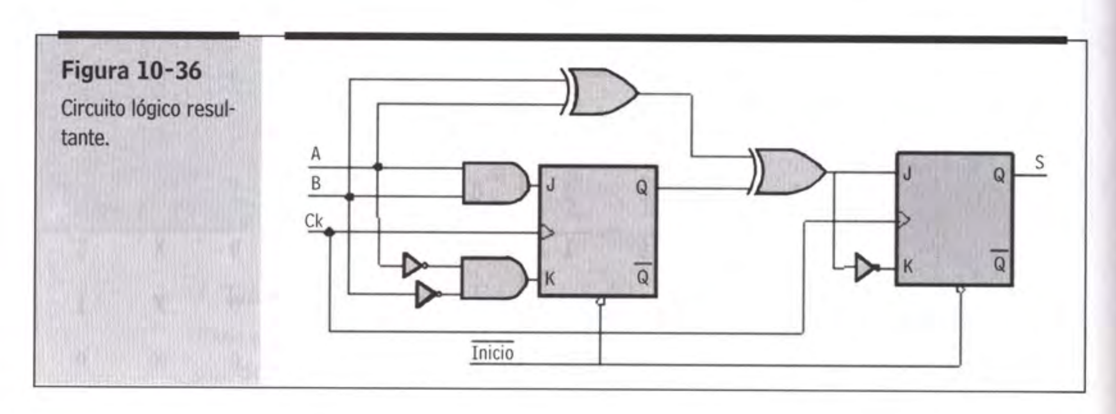


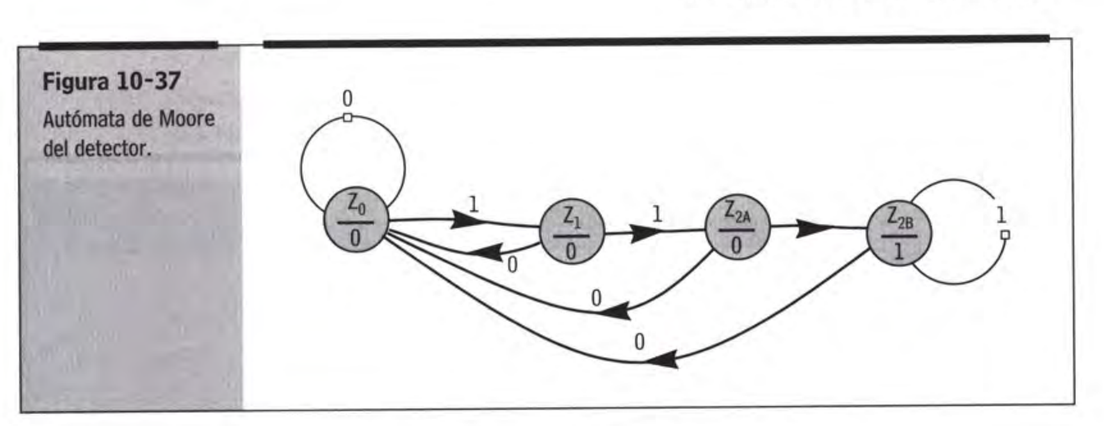
Figura 10-36
Circuito lógico resultante.

Diseñemos ahora el autómata de Moore correspondiente a un detector de tres o más unos.

1. Obtención del DTE

El DTE del autómata es el mostrado en la figura 10-37.

Figura 10-37
Autómata de Moore del detector.



2. Obtención de la Tabla de Transición de Estados y de Salida

Reordenemos el anterior diagrama en la tabla de transición de estados de la tabla 10-20.

Tabla 10-20
Tabla de transición de estados y de salidas.

	A	
	0	1
Z_0	Z_0	Z_1
Z_1	Z_0	Z_{2A}
Z_{2A}	Z_0	Z_{2B}
Z_{2B}	Z_0	Z_{2B}

Est.	S
Z_0	0
Z_1	0
Z_{2A}	0
Z_{2B}	1

3. Tabla de Transición de Estados y Salida Codificada

De nuevo soslayamos el problema de Asignación de Estados, codificando en la tabla 10-21 de una forma tan arbitraria como simple e intuitiva.

Tabla 10-21
Tabla de codificación y de transición de estados codificada.

Estado	Q_1	Q_0	S
Z_0	0	0	0
Z_1	0	1	0
Z_{2A}	1	0	0
Z_{2B}	1	1	1

		A	
Q_1	Q_0	0	1
0	0	00	01
0	1	00	10
1	0	00	11
1	1	00	11

4. Tabla de Excitación de biestables

Planteemos la tabla 10-22 para biestables J-K.

Tabla 10-22

Tabla de excitación del autómata.

t			t+1					
Q_1	Q_0	A	Q_1	Q_0	J_1	K_1	J_0	K_0
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	0	0	0	X	X	1
0	1	1	1	0	1	X	X	1
1	0	0	0	0	X	1	0	X
1	0	1	1	0	X	0	1	X
1	1	0	0	0	X	1	X	1
1	1	1	1	1	X	0	X	0

5. Simplificación de funciones

Planteemos y simplifiquemos los diagramas V-K de la figura 10-38 correspondientes a la anterior tabla y a la salida.

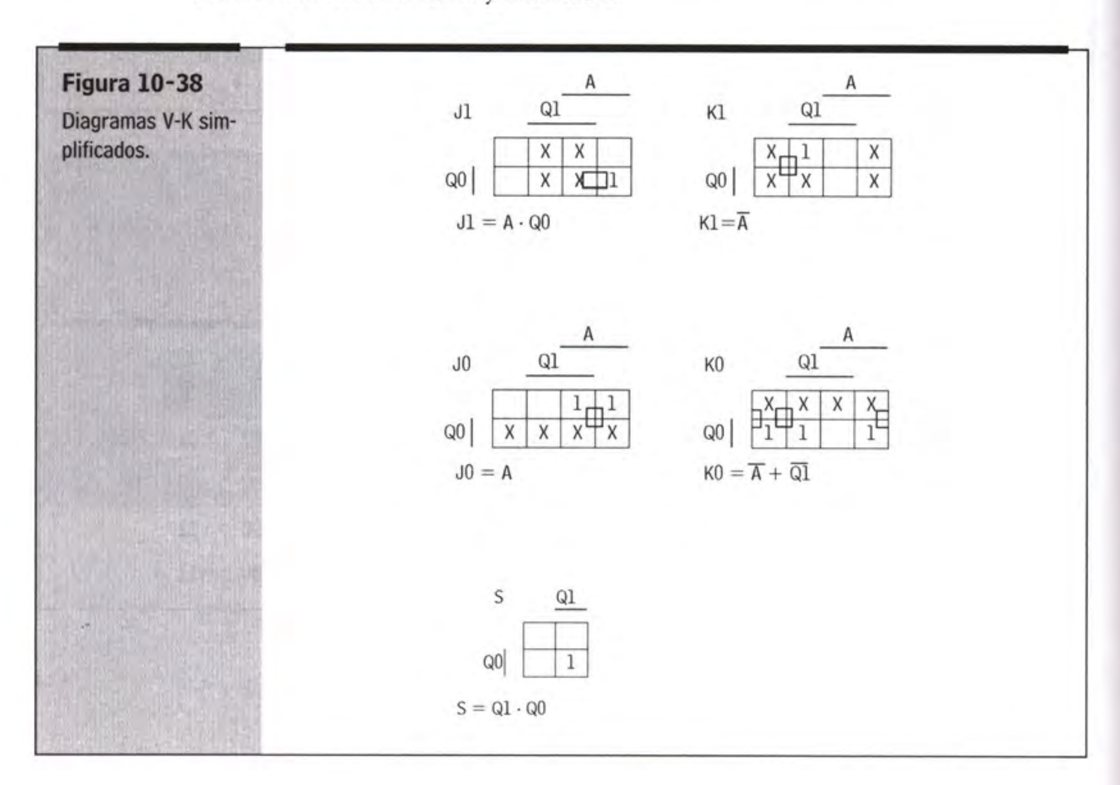


Figura 10-38

Diagramas V-K simplificados.

6. *Implementación del circuito y análisis*

Sólo resta obtener el circuito de la figura 10-39 correspondiente a las anteriores ecuaciones.

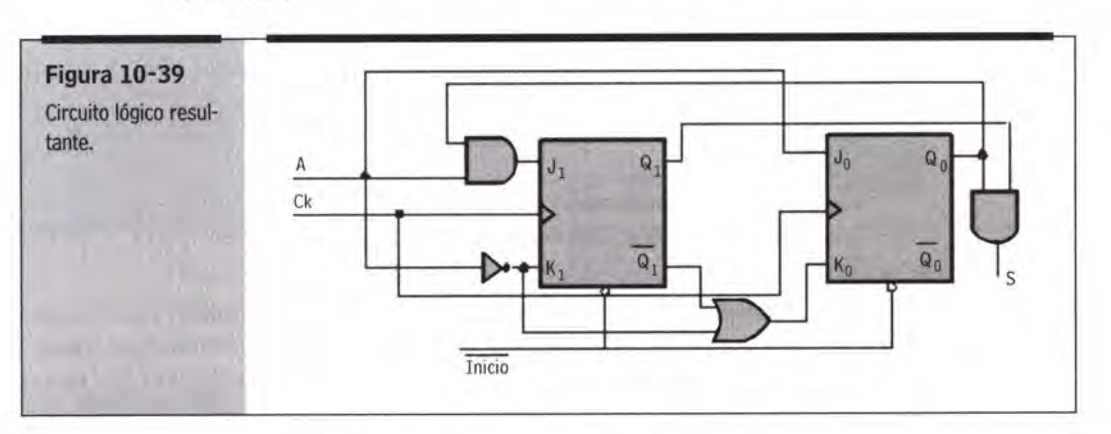


Figura 10-39
Circuito lógico resultante.

Mediante los ejemplos anteriores queda claro cuál es el proceso a seguir para diseñar un autómata de Moore o Mealy. Dicho proceso es sistemático y sencillo, y sólo depende de la calidad del DTE, que a su vez depende de la complejidad del sistema y de la habilidad del diseñador.

10.5. Minimización de estados y codificación de estados

A la hora de diseñar hemos dejado sin completar dos pasos: la minimización de estados y la codificación de estados.

De un autómata sólo nos interesa la salida que va produciendo, no el número de estados que necesita para hacerlo, es decir, nos interesa la salida, no por cuántos estados distintos ha pasado. Por lo tanto, es claro el interés de minimizar la tabla de estados: obtener el mínimo número de estados sin degradar el autómata.

Una vez que tenemos los estados, minimizados o no, resta codificarlos. La primera opción es codificarlos arbitrariamente *en orden,* como en las tablas 10-9, 10-14 y 10-21. Pero resulta que no todas las codificaciones derivan en autómatas del mismo tamaño, es decir, una acertada codificación de estados supone economizar en la implementación.

Ya están definidos los problemas de minimización y codificación de estados, pero su resolución no compete al nivel de este libro. Cabe decir que minimizar supone aplicar un algoritmo tan sencillo como largo y tedioso; que está implementado en la mayoría de los entornos computacionales de diseño digital, incluido el BOOLE-DEUSTO. Por otro lado, para resolver el problema de codificación de estados no se dispone de un algoritmo, sino de una serie de reglas heurísticas de aplicación larga y resultado dudoso.

Al nivel de este libro no le competen la resolución de estos dos problemas. El lector interesado encontrará en la bibliografía clásica suficiente información.

10.6. Implementación de máquinas secuenciales

Durante este capítulo hemos diseñado los autómatas de una forma canónica: tablas, biestables y puertas. Sin embargo, existen otros modos de implementar que parten de enfoques distintos:

- Implementación de autómatas con los estados codificados con el código *one-hot*.
- Implementación desde otros enfoques hardware: contadores, multiplexores, memorias, etc.
- Implementación utilizando técnicas ASM y RT.
- Implementación con dispositivos lógicos programables: PLD, CPLD, FPGA, etc.

Cualquiera que sea la opción elegida, el punto de partida será el mismo que el visto en este capítulo: el DTE. Así, el lector interesado sólo deberá ahondar en el uso que cada una de las técnicas anteriores hace del DTE. Quiza de todas ellas ahora la más común sea la que usa dispositivos lógicos programables.

10.7. Sistemas secuenciales asíncronos

En los autómatas vistos hasta ahora el reloj es tan importante como las entradas, y así se refleja en las técnicas de análisis y diseño. Pero ¿qué ocurre si no hay un reloj? Es decir, si las entradas y salidas evolucionan libremente sin el ritmo de un reloj. De hecho, los sistemas secuenciales asíncronos son muy comunes, por ejemplo:

- Controlar el motor de un limpiaparabrisas con detector de reposo.
- Controlar la apertura de una puerta de entrada con un detector de personas.
- Controlar la activación de las bombas de un depósito con detectores de nivel.

En los anteriores ejemplos las entradas cambian sin un ritmo preestablecido, lo hacen asíncronamente. De hecho, esto es lo más normal en sistemas que podríamos llamar *industriales*. La cuestión es: ¿cómo se puede diseñar un sistema secuencial asíncrono?

Existen dos caminos. En primer lugar, y al igual que para sistemas síncronos, existen técnicas propias de análisis y diseño de sistemas asíncronos. Ahora bien, los sistemas asíncronos no son deseables, ya que su implementación puede dar lugar a problemas de distinto tipo, lo que nos lleva a plantear otra solución: sincronizar los sistemas asíncronos.

La idea es la siguiente: introducir un reloj de frecuencia suficiente cuya misión será *despertar* al sistema en cada flanco para observar sus entradas y decidir sus salidas. Utilizando otra terminología se puede decir que el reloj hace de "*watchdog*", de perro guardián. La figura 10-40 muestra cómo una entrada asíncrona es sincronizada con el flip-flop D antes de ser procesada por el autómata.

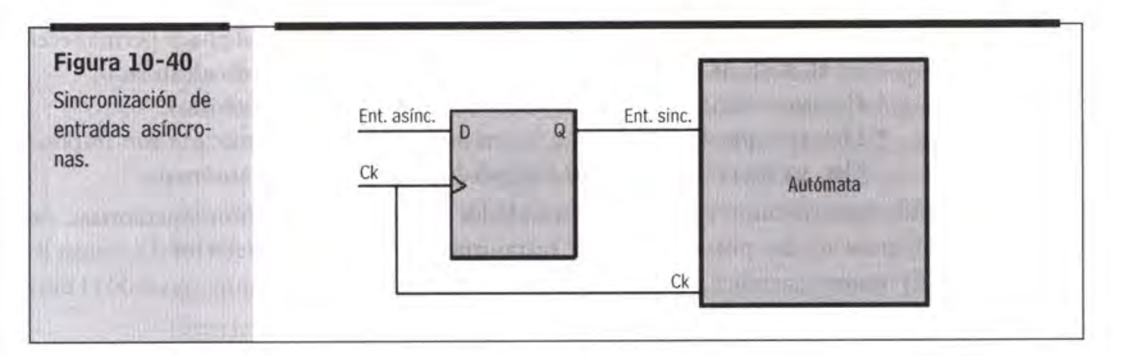


Figura 10-40
Sincronización de entradas asíncronas.

Resolvamos como ejemplo el control del arranque de un motor con dos pulsadores, uno de arranque A y otro de paro P. El autómata resultante es el de la figura 10-41, donde el orden de las entradas en las transiciones es AP (arranque y paro).

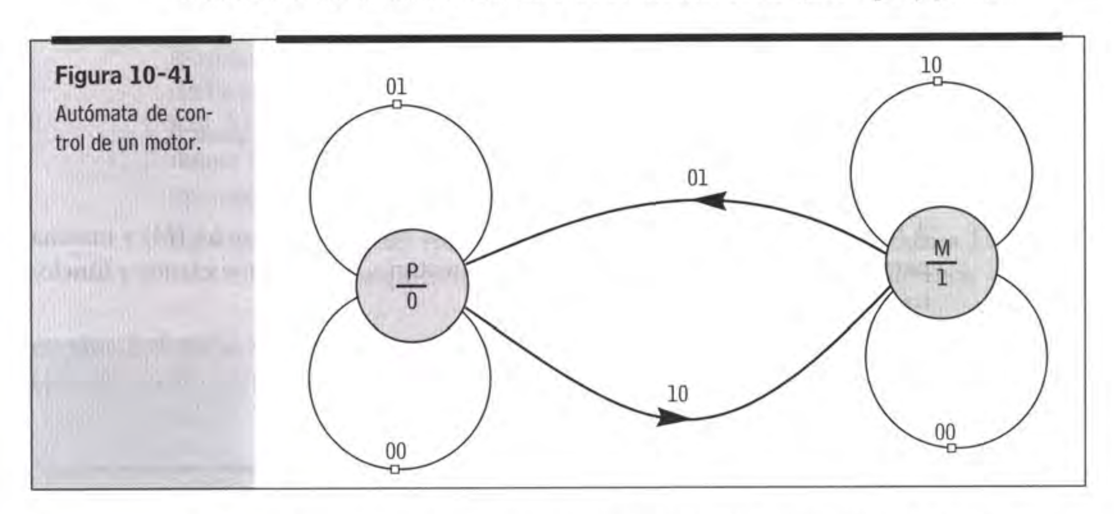


Figura 10-41
Autómata de control de un motor.

Pensemos que el autómata tiene un reloj de 1 KHz (por ejemplo), así el autómata observará cada milisegundo qué valor tienen los pulsadores para decidir si arranca o para el motor. Es cierto que si un cambio en los pulsadores durara menos de 1 ms podría no ser visto por el autómata, pero parece poco probable. En cualquier caso, podríamos aumentar la frecuencia hasta 1 MHz, con lo que ya no habría dudas.

Vemos por tanto que el reloj es añadido al sistema de forma artificial para que el sistema sea síncrono, aunque parezca asíncrono. El planteamiento y resolución de estos autómatas es el ya visto: búsqueda de estados y transiciones, obtención del DTE e implementación del sistema; sin embargo, se pueden dar una serie de consejos específicos:

- Ordenar la *historia del sistema* en transiciones y estados que cuenten el funcionamiento del sistema a controlar.
- De una transición a la siguiente en la historia sólo debe cambiar un bit (es casi imposible que dos señales asíncronas cambien simultáneamente).

- La misma transición que lleva al autómata a un estado le hace permanecer en él, es decir, la transición de llegada *rebota* en el estado alcanzado.
- En estos autómatas son muy comunes los estados transitorios.
- Hay que prestar atención a las transiciones que en principio son imposibles, ya que pueden originar la pérdida de control del autómata.

Resolvamos como ejemplo el controlador del motor de un limpiaparabrisas. Se dispone de dos pulsadores A y P (arranque y paro) y de un detector de reposo R. El motor se arranca con el pulsador A y se para con el P, siempre y cuando el limpiaparabrisas haya alcanzado el reposo (R = 1).

Veamos algo de la historia del limpiaparabrisas:

- Las entradas se ordenan APR (arranque, paro y reposo).
- El motor está en *reposo*: 001.
- Se pulsa arranque (101) y el motor *arranca*.
- Al estar *arrancado* (limpiando) el detector de reposo pasa a 0 (100).
- Una vez *arrancado* soltamos el pulsador: 000.
- El limpia *gira* libremente: 000 y 001.
- Se pulsa P (010), pero *no se detiene* hasta alcanzar el reposo.
- Se suelta el pulsador a la *espera de que pare*: 000.
- El detector de reposo se pone a 1 y el motor se *para*: 001.

A la vista de lo anterior resultan tres estados: reposo (R), marcha (M) y marcha para paro (MP). Este último estado es transitorio, e indica que el motor funciona pero se va a parar.

Lo anterior se puede ordenar según un autómata de Moore o Mealy. La figura 10-42 muestra el autómata de Moore resultante con las entradas ordenadas como APR en las transiciones.

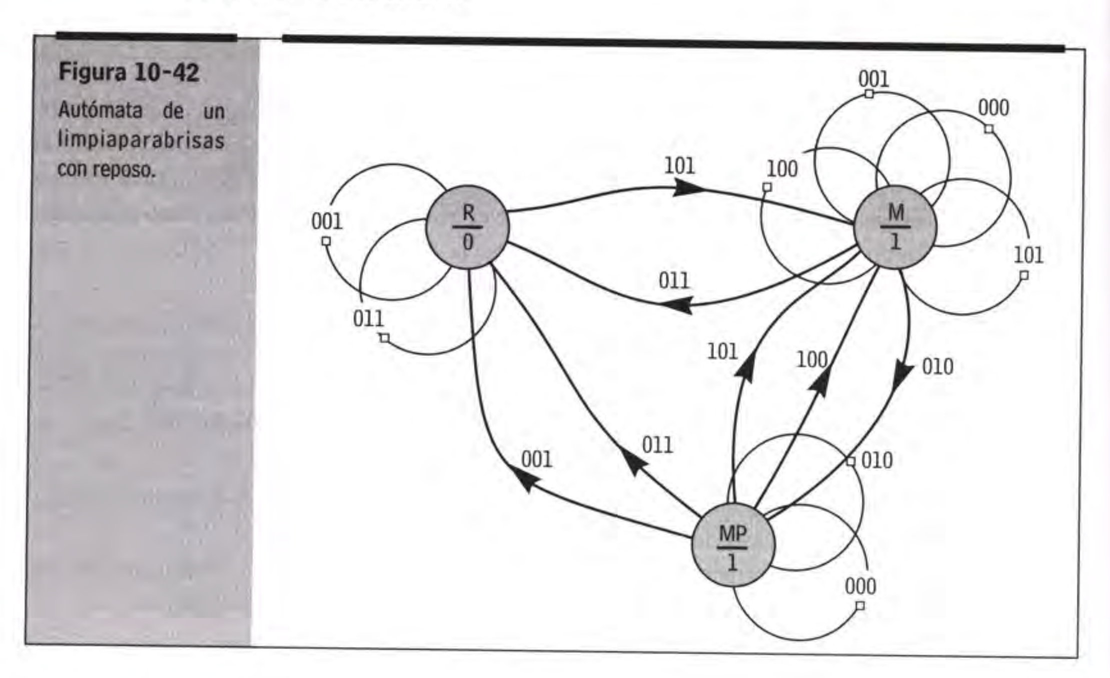


Figura 10-42
Autómata de un limpiaparabrisas con reposo.

En el autómata quizá choque un poco la transición 101 en M. La cuestión es: ¿puede estar arrancado el limpia y estar todavía en reposo? La respuesta dada es sí. Esto no es contradictorio, ya que el sensor puede (y suele) tener un muelle, y éste tardará en retraerse. Además, si no la pusiéramos y se diera, perderíamos el control del autómata y con él el del limpiaparabrisas. Esta situación podría haber sido vista como el rebote de la transición 101 entre los estados R y M, y ya no habría causado problemas.

Además, quedan otras cuestiones. Si las entradas son tres (APR), de cada estado deberían partir 8 transiciones, y no es así: ¿por qué?

En primer lugar, no parece lógico activar A y P simultáneamente. Ahora bien, para preservar el control podemos decir que A = P = 1 será tomado como una orden de paro, es decir, P es prioritario frente a A.

Otra cuestión es: ¿puede estar el motor quieto y no estar en reposo el limpiaparabrisas? La respuesta directa es no; pero bien puede pasar que al arrancar el coche el limpiaparabrisas no esté en reposo (por la razón que sea). En este caso hay dos opciones: dejarlo como está (el conductor decide) o que automáticamente el motor se active hasta llevar el limpia al reposo. Cualquiera de las dos opciones es válida, y de hecho cada coche utiliza una de ellas.

Contemplando todas las opciones, el autómata queda como indica la figura 10-43. Es el lector quien debe analizar el autómata y ver si le convence.

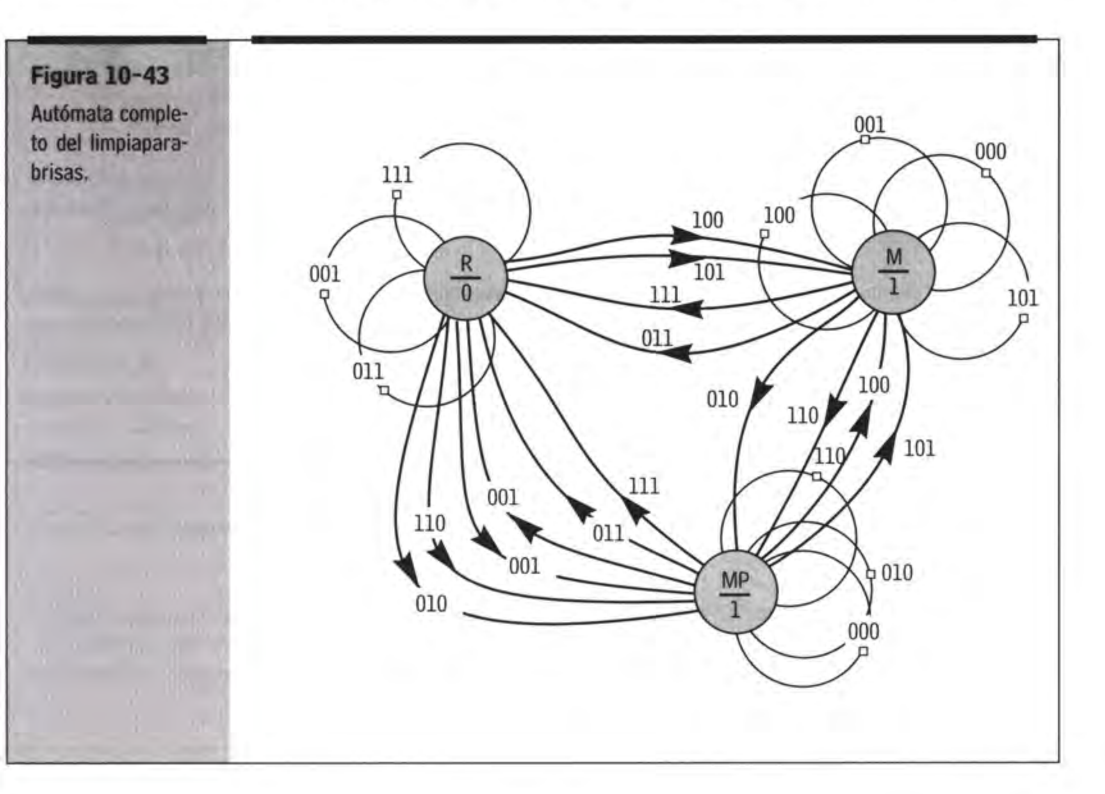


Figura 10-43
Autómata completo del limpiaparabrisas.

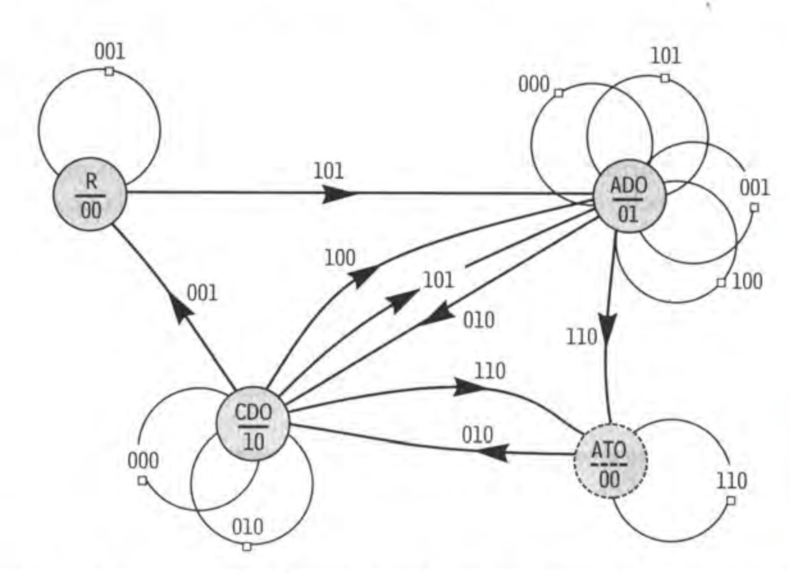
Ejemplo 10-1

Controlar el motor de apertura de una puerta M para que se abra (M = 01), se cierre (M = 10) o se quede quieta (M = 00). Antes de la puerta hay un detector de personas (D); además, la puerta dispone de un detector de abierto (A) y de uno de cerrado (C). Si pasara una persona (D = 1) la puerta completaría un ciclo de apertura-cierre. Si durante el cierre apareciera una persona, la puerta volvería a abrirse. Si al alcanzar la apertura siguiera pasando gente, la puerta quedaría abierta.

La figura 10-44 muestra el autómata de Moore que controla el motor de la puerta. En este autómata no se han anotado las transiciones en principio *absurdas*, quedando esta parte como trabajo del lector. Las entradas se ordenan en las transiciones como DAC (detector, abierto y cerrado).

Figura 10-44

Autómata de control de una puerta con detector de personas.



Véase que no hay tres estados, sino cuatro. En reposo, abriendo, cerrando y abierto. Es interesante que el lector se fije en los participios y los gerundios. Cuando se diseña un autómata de este tipo, ésa es una buena pista.

Por último, se le plantea al lector que piense en puertas de garaje. La situación es muy parecida a la planteada, pero ¿qué ocurre en un garaje si mientras se está cerrando la puerta un nuevo conductor pide que se abra?, ¿completa el ciclo o vuelve directamente a abrir la puerta?

Ejemplo 10-2

Controlar la activación de un motor (M) que dispone de dos pulsadores A y P, de arranque y paro. El arranque o paro no se producirá al pulsar A o P, sino al soltar el pulsador.

El autómata de la figura 10-45 es muy parecido al del limpiaparabrisas. Pero en este caso no arranca al pulsar, sino al soltar. Esto nos lleva a considerar dos estados transitorios: *paro para marcha* (PM) y *marcha para paro* (MP). Queda para el lector el intentar plantear el autómata con dos estados, viendo lo inútil del trabajo. Las entradas son AP (arranque y paro).

Ejemplo 10-2 (Cont.)

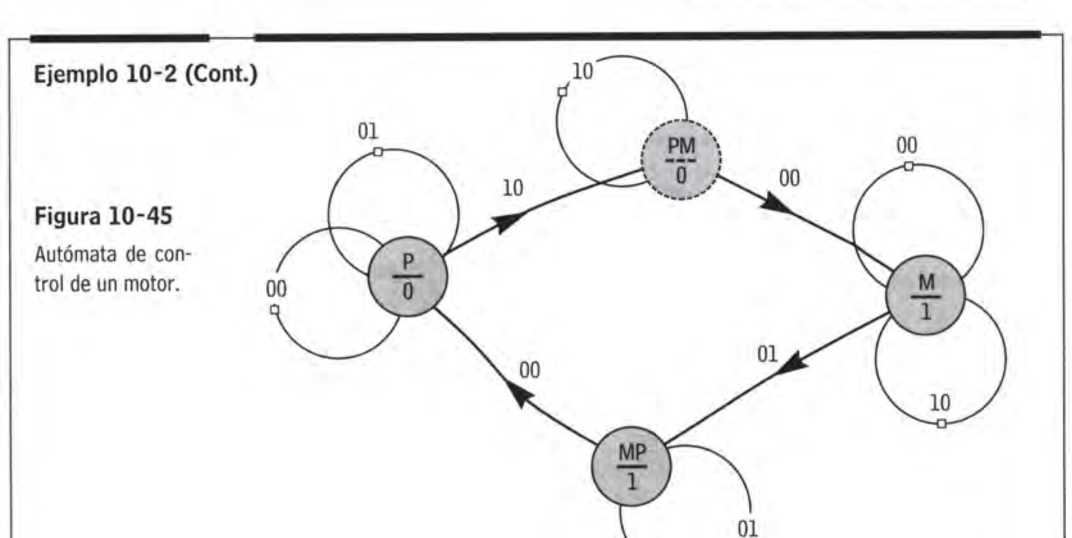


Figura 10-45

Autómata de control de un motor.

Destaquemos una situación: estando en marcha (M), ¿no deberíamos ir con 10 a PM, en vez de a M? La respuesta es no, porque eso conllevaría una absurda parada momentánea del motor. La situación es algo rara ya que responde a estando en marcha (M) el usuario activa marcha (10). No parece normal hacerlo, pero en cualquier caso, y a nuestro entender, no se debe parar el motor, por muy simétrico que quede.

Ejemplo 10-3

Controlar la activación de dos bombas (B_1 y B_2) que deben mantener el nivel de agua de un depósito que dispone de dos sensores: inferior y superior (I y S). Si el agua ha superado un sensor, éste se pone a 1. Si el depósito estuviera lleno (I = S = 1) no se activaría ninguna bomba. Si el depósito estuviera vacío (I = S = 0) se activarían ambas bombas. Si el depósito estuviera mediado (I = 1 y S = 0) se activaría una única bomba: la última en no activarse, es decir, las bombas se alternan en su trabajo.

En los autómatas de las figuras 10-46 y 10-47 las entradas se ordenan en las transiciones como SI (superior e inferior).

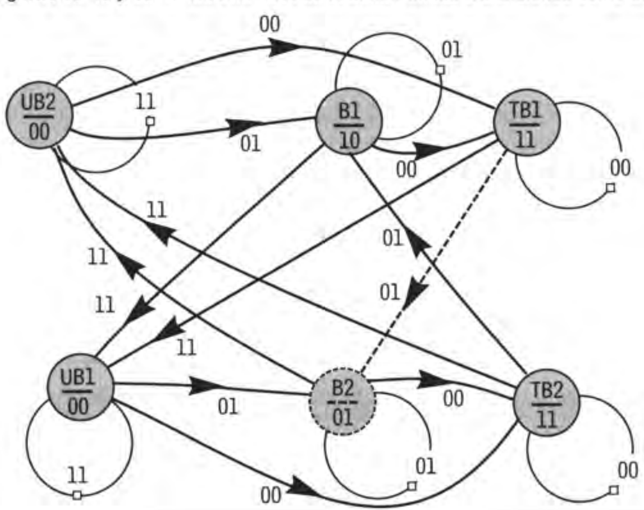
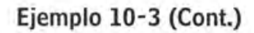


Figura 10-46

Autómata de Moore de control de bombas.



Ejemplo 10-3 (Cont.)

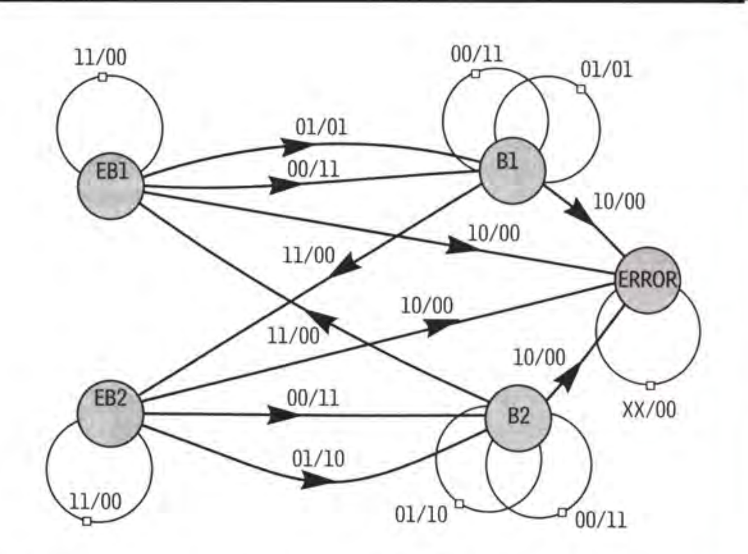


Figura 10-47
Autómata de Mealy de control de bombas.

En este caso se han planteado los autómatas de Moore (figura 10-46) y Mealy (figura 10-47), y en el segundo se incluye un estado de error (E) cuando $I = 0$ y $S = 1$ (hay agua arriba pero no abajo). En este ejemplo vuelven a manifestarse las diferencias entre Moore y Mealy, aunque queda una pregunta: ¿es igual el régimen de trabajo de las dos bombas para los dos autómatas? ¿En qué situación se comportan de distinta forma? ¿Cuál es mejor autómata?

10.8. Metaestabilidad

En el punto anterior hemos visto cómo sincronizar entradas asíncronas para poder diseñar los correspondientes sistemas como autómatas, pero existe un problema a considerar: la metaestabilidad.

Empecemos diciendo que la metaestabilidad es un concepto y problema no exclusivo de los sistemas asíncronos; más bien es un problema de los biestables síncronos por flanco: los flip-flop.

En realidad, la salida de un biestable puede ser uno de tres: 1, 0 o metaestable. Cuando la salida de un flip-flop es metaestable, resulta que no es ni 1 ni 0, y peor aún, puede ser visto por algunos elementos del sistema como 1 y por otros como 0. Claramente es una situación indeseable.

La salida queda metaestable cuando no se respeta el tiempo de set-up exigido por los fabricantes de CI's de flip-flops. Es decir, la entrada asíncrona tiene que ser estable el tiempo suficiente previo al flanco del reloj (t_{set-up}); en caso contrario la salida puede quedar metaestable. Pero resulta que como la entrada es asíncrona, ésta cambiará en cualquier momento, sin atender a tiempos de set-up.

La siguiente pregunta es: ¿cuánto tiempo queda metaestable la salida? En principio, la respuesta es que este tiempo no es predecible a priori. Es más, aunque la probabilidad de que la salida sea metaestable decrece con el tiempo, ésta nunca será 0.

Para evitar la metaestabilidad con determinada probabilidad basta con utilizar la solución de la figura 10-48.

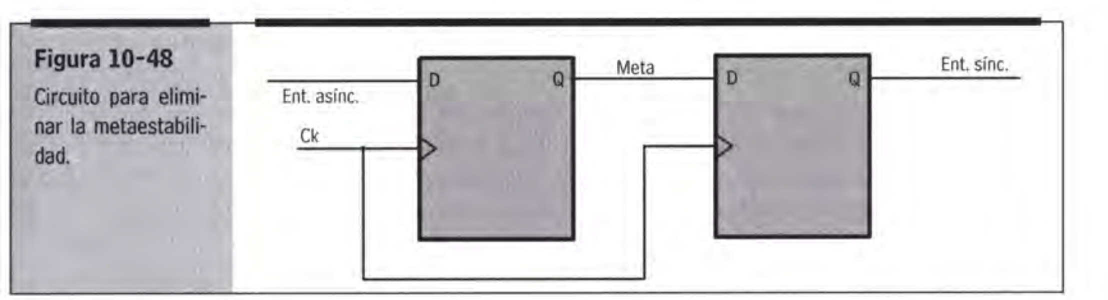


Figura 10-48
Circuito para eliminar la metaestabilidad.

Al introducir un nuevo flip-flop se pretende que aunque META sea metaestable, este estado desaparezca antes del nuevo flanco que asociará META a la ENT. SINC. Es decir, el segundo f-f *no deja pasar* la metaestabilidad hasta el siguiente flanco del reloj; momento en el cual esperamos que la señal ya no sea metaestable. Un efecto de este circuito es que el sistema se ha retardado un pulso respecto del original.

La idea anterior es muy utilizada, y preserva a los sistemas relativamente bien de la metaestabilidad, pero no garantiza su eliminación. Existen dos formas de mejorar el rendimiento del sistema:

- Reducir la frecuencia del reloj y así aumentar el tiempo para que desaparezca la metaestabilidad. Esto implicaría ralentizar el sistema.
- Replicar el circuito añadiendo nuevos flip-flops. La probabilidad de metaestabilidad se verá reducida, y con ella el sistema se verá retardado.

En ambos casos, la solución dada degradaría el comportamiento del sistema.

Una segunda cuestión es: ¿que probabilidad tiene una señal de quedar metaestable? La forma de expresar esta medida es calculando el tiempo medio entre fallos (MTBF en los textos). Por ejemplo, si para un circuito resultara un valor de 100 siglos querría decir que habría un fallo cada 100 siglos. Pero si vendiéramos 10.000 unidades del circuito, se daría un fallo cada año. El cálculo de MTBF se basa en observaciones prácticas y excede a los objetivos de este libro.

Resumiendo, la metaestabilidad no debe ser olvidada en sistemas complejos y/o de elevada frecuencia y rendimiento, pero no debe preocupar mucho en sistemas con necesidades más relajadas. En cualquier caso, el circuito propuesto en la figura 10-48 es en principio una solución adecuada al problema.

La metaestabilidad es un campo complejo y de gran interés teórico, pero, más allá de su descripción, queda fuera de los objetivos de este libro.

10.9. Limitaciones de la máquina de estados finitos determinista

En este capítulo hemos atendido al diseño de máquinas de estados finitos y deterministas. El hecho de que éstos sean deterministas no es una gran restricción, ya que en principio a la industria no le interesan diseños no deterministas.

Por lo visto en el capítulo se puede deducir que de un correcto diseño de un sistema secuencial se puede llegar a implementar cualquier funcionalidad, entre ellas un computador. Sin embargo, la realidad no es ésta.

Veamos otra restricción de las máquinas estudiadas en este capítulo: las máquinas deben plantearse con un número de estados finitos. En principio, esta restricción no parece importante, puesto que con un número finito de estados se puede tener una memoria infinita, y por lo tanto contemplar cualquier diseño por grande que sea sin necesidad de memoria. Pero esto no es cierto para todos los sistemas: para algunos sí, pero para otros no. Por ejemplo, un autómata de los vistos puede detectar tres o más unos (o cualquier secuencia fijada) en una secuencia infinita de entrada. Un autómata también puede sumar dos secuencias de entrada infinitamente largas. Pero ¿puede un autómata multiplicar dos secuencias infinitas? La respuesta es no.

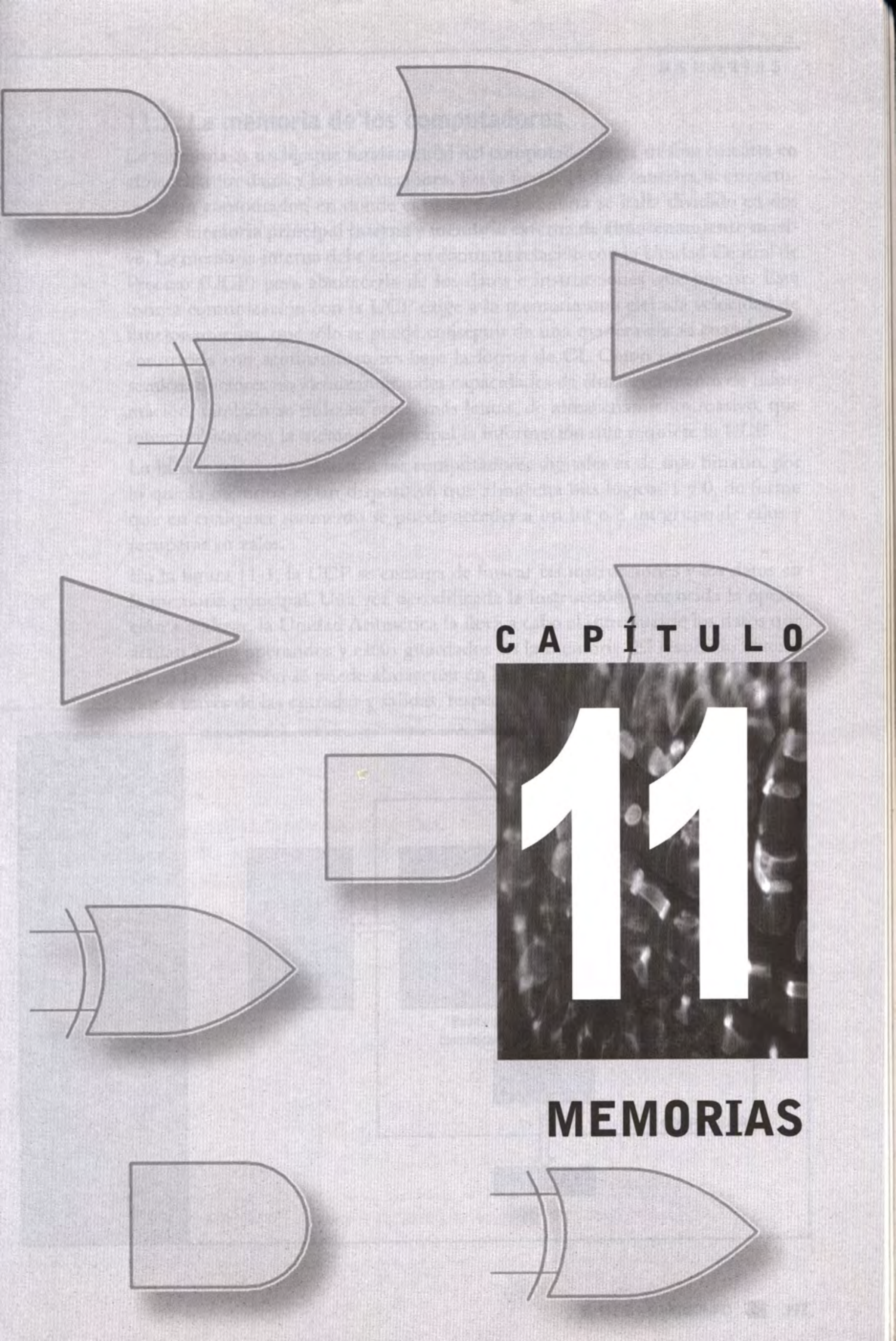
Un autómata con estados finitos puede multiplicar 3 x 3, 4 x 4 o 100 x 100 bits, pero no puede plantearse el autómata para multiplicar n x n bits. La respuesta teórica a este problema fue planteada por Turing, y si intentamos plantear el autómata veremos que no es posible. A otros muchos ejemplos les pasa lo mismo.

La máquina de estados infinitos o máquina universal, también llamada máquina de Turing, puede implementar y resolver cualquier algoritmo. Un computador no es otra cosa que una máquina de Turing reducida, donde el software con sus estructuras de datos y de programa cumple la función del diagrama de transición de estados. Con una máquina programable las posibilidades superan las limitaciones propias de las máquinas con estados finitos.

10.10. Resumen

En este capítulo hemos abordado el análisis y diseño de autómatas finitos deterministas mediante métodos completamente descritos. Además, hemos planteado y resuelto, en cierta medida, el diseño de sistemas secuenciales asíncronos mediante su sincronización.

Al acabar este capítulo el lector debe tener claro qué es un autómata, y qué diferencias hay entre Mealy y Moore. Sabiendo elegir en cada caso la estrategia que mejor se adecue a las necesidades de su sistema.



CAPÍTULO 11

MEMORIAS

11.1. La memoria de los computadores

La memoria es un bloque fundamental del computador, cuya misión consiste en almacenar los datos y las instrucciones. En la figura 11-1 se muestra la estructura de un computador, en donde el bloque de memoria se halla dividido en dos partes: **memoria principal interna y memoria externa de almacenamiento masivo.** La memoria interna debe estar en continua relación con la Unidad Central de Proceso (UCP) para abastecerla de los datos e instrucciones que precise. Esta íntima comunicación con la UCP exige a la memoria una elevada velocidad de funcionamiento, que sólo se puede conseguir de una manera eficaz cuando está construida con semiconductores bajo la forma de CI. Como las memorias de semiconductores no alcanzan grandes capacidades de almacenamiento de información, también se utilizan otras, más lentas, de almacenamiento masivo, que intercambian con la memoria principal la información que requiere la UCP.

La información que manejan los computadores digitales es de tipo binario, por lo que la memoria es un dispositivo que almacena bits lógicos 1 y 0, de forma que en cualquier momento se puede acceder a un bit o a un grupo de ellos y recuperar su valor.

En la figura 11-1, la UCP se encarga de buscar las instrucciones y los datos en la memoria principal. Una vez decodificada la instrucción y conocida la operación a realizar, la Unidad Aritmética la lleva a cabo al introducirle los datos que actúan como operandos y están guardados en la memoria. El resultado obtenido en la operación se puede almacenar en la memoria o sacarlo al mundo exterior a través de las entradas y salidas, respectivamente.

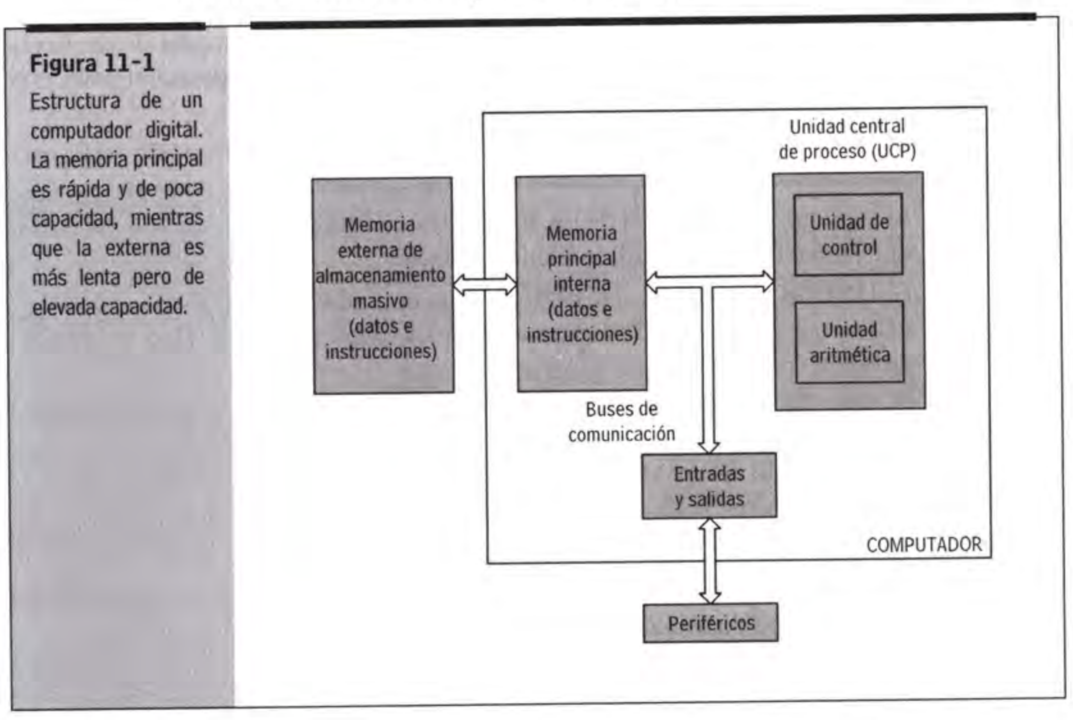


Figura 11-1
Estructura de un computador digital. La memoria principal es rápida y de poca capacidad, mientras que la externa es más lenta pero de elevada capacidad.

Frecuentemente, la memoria principal no dispone de suficiente capacidad para contener todos los datos e instrucciones, en cuyo caso se necesitan otras memorias auxiliares que funcionan como periféricos del sistema y cuya información se transfiere a la memoria principal cuando es preciso. Las memorias externas son más lentas que la principal, pero poseen una capacidad de almacenamiento enorme.

Las **celdas** de las memorias sólo pueden almacenar un bit 1 o 0. Se denomina **punto de información** al elemento de la memoria que es capaz de almacenar un bit.

La implementación física de un punto de memoria es muy diversa y depende de la tecnología de fabricación. Así, existen puntos de memoria implementados como flip-flops con semiconductores, otros son almacenados en discos y cintas en los que se magnetizan pequeñas áreas de su superficie y también hay puntos de memoria sin una posición fija sobre el soporte físico, desplazándose a lo largo del mismo, como sucede en los dispositivos de burbujas magnéticas.

Para almacenar y recuperar un bit de un punto de memoria se requiere una combinación de las siguientes señales:

1ª. *Señales de direccionamiento*. Seleccionan o determinan la posición de un bit de memoria.
2ª. *Orden de lectura o escritura*. Elige la operación que se va a realizar.
3ª. *Señal de reloj*. Sincroniza la entrada y salida del bit en la memoria.

11.1.1. Evolución histórica

En las calculadoras de la década de los 30 se emplearon las **tarjetas perforadas**, como memorias. La dirección de las posiciones a acceder quedaba determinada por la situación de las ruedas dentadas de arrastre de la tarjeta. Un poco más tarde se pasó a emplear **relés electromagnéticos**.

En 1946, el computador **ENIAC** utilizaba **válvulas electrónicas de vacío** para construir los biestables que conformaban los puntos de memoria.

Al comienzo de la década de los 50 se usaron las **líneas de retardo de mercurio**, que tenían una capacidad de 1 kbit por línea.

El **UNIVAC I** introdujo, en 1951, la primera unidad comercial de **banda magnética**, que disponía de una capacidad de 1,44 Mbit y una velocidad de 100 pulgada/s.

El primer computador comercial que utilizó como memoria principal el **tambor magnético** fue el **IBM 650** en 1954. Dicho tambor giraba a 12.500 rpm y tenía una capacidad de 120 kbit.

En 1953, el **MIT** dispuso de la primera memoria operativa de **ferritas**, que se hizo popular hasta mediados de los años 70.

IBM, en 1968, diseñó la primera memoria comercial de **semiconductores**. Tenía una capacidad de 64 bits.

El modelo 350 de **IBM** fue el primero que empleó un **disco con brazo móvil y cabeza flotante**. Su capacidad era de 40 Mbits y su tiempo de acceso, de 500 ms.


BIBLIOTECA CADIZ PUBLICA

Tecnologías recientes, como las de **burbujas magnéticas, efecto Josephson, acoplamiento de carga, de tipo óptico** y otras, compiten en la actualidad por desplazar a las memorias de semiconductores, basadas en silicio, que alcanzan capacidades de varios Mbit en una pastilla, con un reducidísimo tiempo de acceso y un coste razonable.

En Electrónica Digital, las memorias son los elementos que más rápidamente se desarrollan y cambian. Los espectaculares avances tecnológicos provocan una constante mejora en la capacidad y la velocidad de los dispositivos de memoria, que permiten su adaptación a los computadores modernos.

11.1.2. Niveles de jerarquía

La memoria principal abastece a la UCP de instrucciones y datos. Como la UCP está construida con circuitos integrados rapidísimos, obliga a que la memoria deba emplear un tiempo mínimo en grabar u obtener información de sus celdas. Sin embargo, las memorias rápidas son de pequeña capacidad, por lo que existe en el computador un **nivel** veloz (pero de poco tamaño) que actúa como memoria principal y niveles sucesivos de menor rapidez y mayor capacidad.

La información se deposita en uno de los niveles de memoria, de acuerdo a su prioridad de uso. Así, cuando un programa o unos datos de archivo son poco empleados, se almacenan en un **nivel inferior** más lento y de mayor capacidad; si se necesitan en un momento determinado, se trasladan al **nivel superior** más rápido.

Además de las dos propiedades principales que se han comentado de las memorias (velocidad o tiempo de acceso a la información y capacidad de almacenamiento), hay otra característica que influye en la determinación de su **nivel jerárquico**: el **coste/bit**. Estas tres características son contrapuestas, es decir, que si se desea un tiempo de acceso muy bajo, el coste/bit será alto y la capacidad o tamaño será bajo, tal como se refleja en las figuras 11-2 y 11-3.

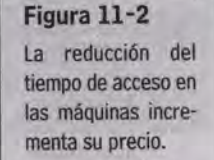
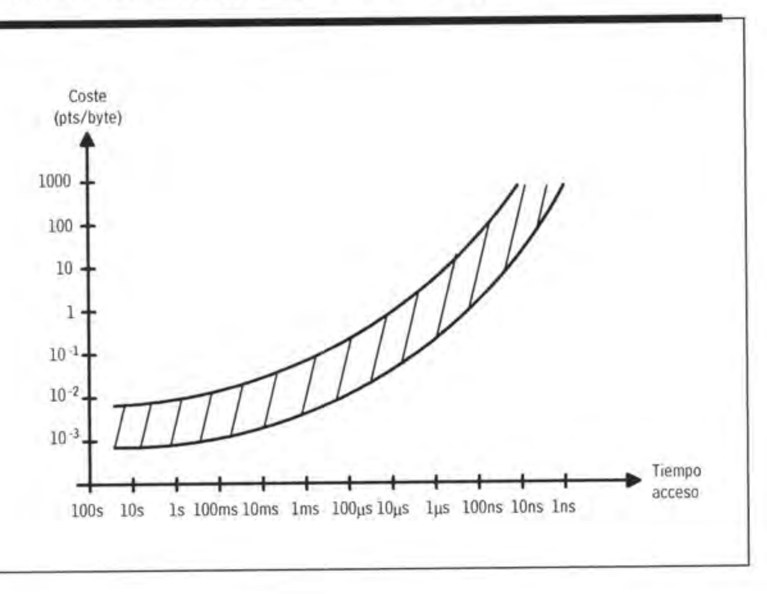


Figura 11-2

La reducción del tiempo de acceso en las máquinas incrementa su precio.



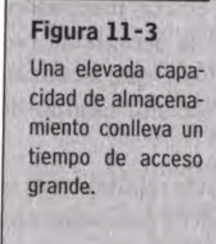
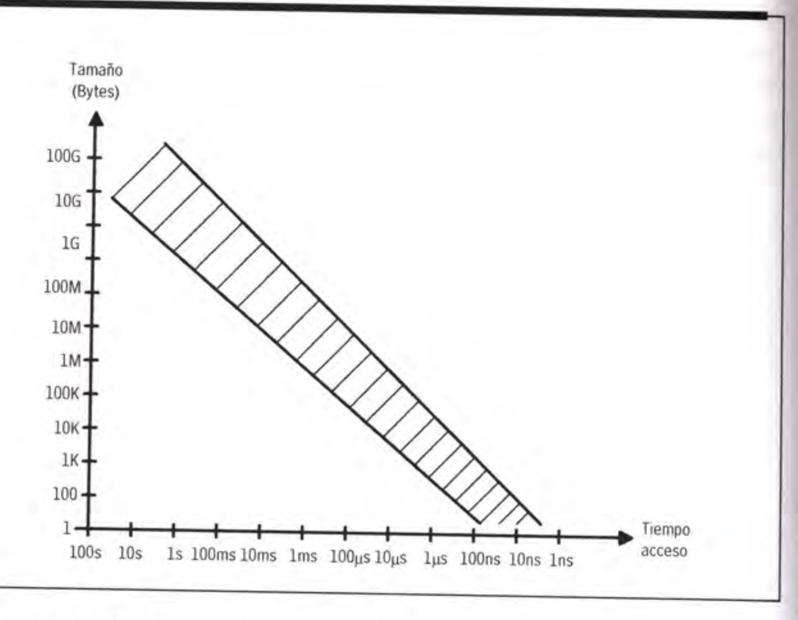


Figura 11-3

Una elevada capacidad de almacenamiento conlleva un tiempo de acceso grande.



La memoria ideal debe poseer una capacidad elevada junto a un tiempo de acceso muy pequeño y un precio reducido. Dichas características, por ahora, son tecnológica y económicamente irreconciliables.

Hay que recurrir a una especialización usando memorias muy rápidas y de pequeña capacidad en aquellas partes del sistema en las que predomine el factor velocidad. Las memorias lentas y de gran capacidad se destinarán a las partes donde prime o sobresalga el factor capacidad.

En la siguiente tabla se presenta una clasificación jerárquica de la memoria junto a sus características más relevantes.

Tabla 11-1

Tipos de memoria.

NIVEL DE JERARQUÍA	CAPACIDAD (BYTES)	VELOCIDAD	ACCESO
REGISTROS	6 - 100	10 - 20 ns	ALEATORIO
MEMORIA CACHÉ	1 K - 512 K	10 - 100 ns	ALEATORIO
MEMORIA PRINCIPAL	1 M - 32 M	20 - 200 ns	ALEATORIO
MEMORIA SECUNDARIA	1 M - 1 G	1 - 100 ns	DIRECTO
MEMORIA AUXILIAR DE DISCO	50 M - 10 G	100 ns	DIRECTO
MEMORIA AUXILIAR DE CINTA	ILIMITADO	MINUTOS	SECUENCIAL

Se pueden distinguir los siguientes niveles jerárquicos:

1º Memorias caché o tampón.

Son de baja capacidad, muy rápidas y construidas con semiconductores. Sus tiempos de acceso oscilan entre unos pocos ns.

2° Memoria principal.

Capacidad no muy alta, tiempos de acceso comprendidos entre unas pocas decenas de ns y construidas, generalmente, con semiconductores.

3° Memorias intermedias.

Suelen ser de acceso secuencial, aleatorio o híbridas, con tiempo de acceso del orden de ms y una elevada capacidad, medida en Gigabytes (miles de millones de bytes).

Este nivel suele ser sustituido por el de **memoria secundaria** (discos de cabeza fija) y por el nivel de **memoria auxiliar**, que incluye los discos móviles y las cintas.

4° Memorias auxiliares.

Son lentas y de gran capacidad. Se pueden citar, como ejemplos representativos, los discos de cabeza móvil y las cintas magnéticas.

En la figura 11-4 se presenta un gráfico que describe, en forma simplificada, la jerarquía de los diversos tipos de memoria.

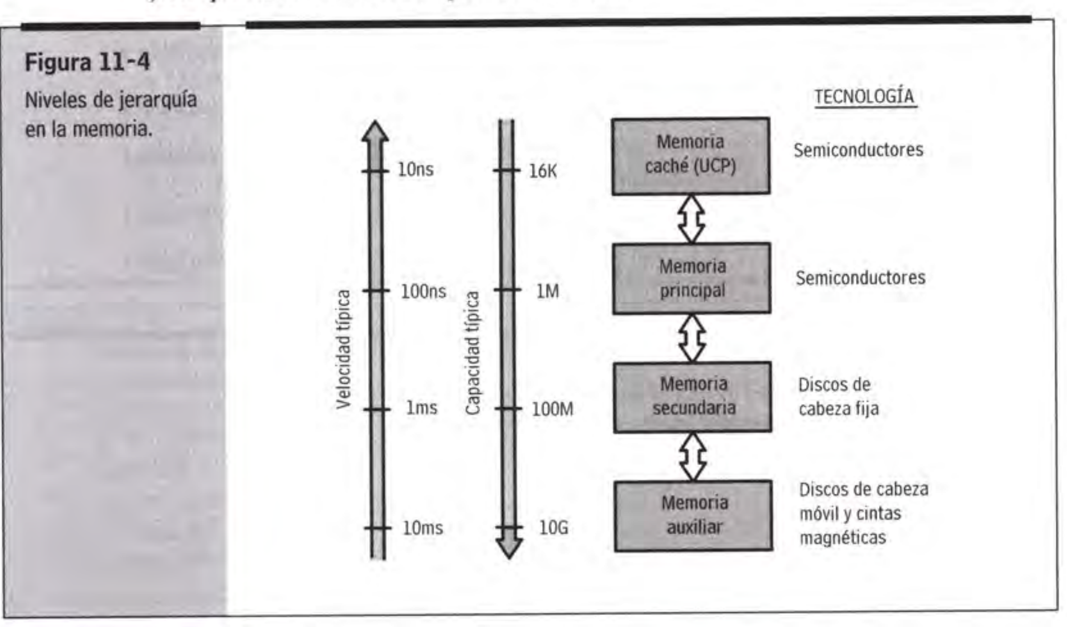


Figura 11-4
Niveles de jerarquía en la memoria.

11.2. Características generales

Una memoria consiste en un conjunto de registros, cada uno de los cuales almacena varios bits. Cada registro o posición de la memoria guarda unos bits que constituyen un dato o una instrucción del computador y reciben el nombre de palabra. Normalmente, el tamaño de las palabras que emplean los computadores está comprendido entre 4 y 64 bits.

Cada palabra ocupa una posición de memoria a la cual se la referencia con una **dirección**. Así, cuando una memoria está formada por ocho palabras de 4 bits, su estructura interna puede representarse como se refleja en la figura 11-5, en donde cada palabra queda especificada por una dirección.

En general, el tamaño de las palabras es un múltiplo de 8 bits. Recibe el nombre de **byte** el conjunto de 8 bits. Cuando se hace mención de la capacidad de una memoria, la unidad de medida suele ser el byte, aunque a veces también se expresa en bits. Para no confundir ambas unidades, la primera se representa abreviadamente con **B : byte** y la segunda con **b : bit**. De esta forma, 64 B significa 64 bytes, mientras que 16 b representa 16 bits. En el caso de la memoria de la figura 11-5, el tamaño podía indicarse como de 8 x 4 b, es decir, ocho palabras de 4 bits.

Figura 11-5
Estructura interna de una memoria formada por ocho palabras de 4 bits cada una.

DIRECCIÓN	BIT 3	BIT 2	BIT 1	BIT 0	
0 0 0					PALABRA 0
0 0 1					PALABRA 1
0 1 0					PALABRA 2
0 1 1					PALABRA 3
1 0 0					PALABRA 4
1 0 1					PALABRA 5
1 1 0					PALABRA 6
1 1 1					PALABRA 7

(CONTENIDO: BIT 3 – BIT 0)

Ejemplo 11-1

a) Dibujar la estructura interna de una memoria de cuatro palabras de 16 bits.
b) Indicar su capacidad total en bits.
c) Indicar su capacidad total en bytes.

SOLUCIÓN

a) Véase la figura 11-6.

Figura 11-6
Estructura de una memoria de cuatro palabras de 16 bits

DIRECCIÓN — CONTENIDO (B15 ... B0)

DIRECCIÓN	B15 … B0	
0 0		PALABRA 0
0 1		PALABRA 1
1 0		PALABRA 2
1 1		PALABRA 3

b) 64 bits.
c) 8 bytes.

Para realizar una operación de lectura o escritura en la memoria, hay que especificar, además del tipo de operación deseado, la dirección de la posición a la que hay que acceder. También existe una señal de reloj que sincroniza la entrada y la salida de información en la memoria.

A continuación, se definen las características más importantes de las memorias, para que sirvan de elementos de comparación entre los diversos tipos.

CAPACIDAD

Es el número total de bits que puede almacenar una memoria. Dicho número debe ser una potencia de 2, puesto que el número de posiciones que pueden existir se referencian con direcciones de **n** bits, lo que significa que la cantidad será 2^n. Por otra parte, las palabras tienen un tamaño que es potencia de 2, desde $2^2 = 4$ hasta $2^7 = 64$ bits.

La capacidad se suele expresar por el producto **palabras x número de bits por palabra**. Así, la capacidad de una memoria de 1.024 palabras de 32 bits cada una se expresa como 1.024 x 32.

Aunque las memorias sólo pueden tener como número de posiciones más próximo a 1.000 el valor $2^{10} = 1.024$, para abreviar esta cantidad se define **1K = 1.024**, cuando se habla de memorias. De esta forma, 8 K = 8.192 y 16 K = 16.384. Igualmente, el número de posiciones que puede existir más próximo al millón es 2^{20} = 1.048.576, cantidad que se representa abreviadamente por **1 M (1 Mega)**. Por último, con la abreviatura G se representan los "Gigas", teniendo en cuenta que **1 G = 1.024 M.**

Ejemplo 11-2

Se dispone de un CI, modelo 2732, que funciona como una memoria de 4K x 8 de capacidad. Se pide,

a) ¿De cuántos bytes dispone?

b) ¿Cuántos bits tiene en total?

c) ¿De cuántos bits se compone la dirección de cada posición de la memoria?

SOLUCIÓN

a) 4K = 4.096 bytes.

b) 4.096 x 8 = 32.768 bits.

c) Como 4.096 = 2^{12}, cada dirección se compondrá de 12 bits.

DIRECCIONAMIENTO

Se llama **dirección** a un número binario que sirve para definir una posición concreta de la memoria. Existirán tantas direcciones como posiciones, por lo tanto, si hay 2^n posiciones, la dirección se compone de **n** bits, ya que con dicho número de bits se pueden realizar 2^n combinaciones diferentes. Por ejemplo, en una memoria que consta de 1 K posiciones, la dirección está formada por 10 bits, puesto que 2^{10} = 1 K. La primera dirección se compondrá de 10 ceros, mientras que la última tendrá 10 unos.

Ejemplo 11-3

Se dispone de una memoria de 16 palabras de 8 bits cada una.

a) ¿Cuántos bits tiene la dirección de cada palabra?

b) Si la primera palabra de la memoria es la **palabra 0** y la última la **palabra 15**, indicar la dirección de la **palabra 9** en binario.

c) ¿Cómo afecta al número de bits de la dirección el tamaño de cada palabra?

SOLUCIÓN

a) Como $2^4 = 16$, el número de bits de la dirección es 4.

b) 1001.

c) El tamaño de la palabra no influye en el número de bits que forman la dirección de la palabra.

MECANISMO DE DIRECCIONAMIENTO

Se encarga de determinar la posición de memoria a acceder.

En una memoria estática el mecanismo de direccionamiento es inherente a su propia construcción. El conexionado de los transductores, junto a la selección de estos últimos, especifica, de forma inequívoca, el punto de memoria al que se accede. Por este motivo, este tipo de memoria recibe la denominación de **memoria de direccionamiento cableado**.

Por el contrario, en las memorias dinámicas, al estar compartidos los transductores, no existe la mencionada relación. La selección se consigue mediante la Unidad de Control, que interpreta una información adicional, que se llama *información de direccionamiento,* y que se almacena junto a los datos.

A continuación se describen los diversos tipos de direccionamiento.

1°) Direccionamiento cableado.

El mecanismo de direccionamiento de una memoria estática ofrece dos formas típicas de implementación, **2D** y **3D**. Para analizarlas, se considera que la memoria tiene 2^m palabras de n bits cada una. Se trata de acceder simultáneamente a los **n** bits de una palabra, cuya dirección queda definida con **m** bits.

Direccionamiento 2D

Tal como se muestra en la figura 11-7, en este direccionamiento todos los bits de la misma posición en cada palabra (bits 0, 1, ...) están conectados a la misma pareja de transductores. Habrá n parejas de transductores para la lectura y la escritura.

Para seleccionar la palabra deseada (figura 11-7) se decodifican los m bits de dirección en un decodificador m x 2^m, que tiene una señal de salida individualizada para cada palabra de memoria.

Se usa la misma conexión para la lectura que para la escritura, bastando activar el transductor correspondiente para definir la operación.

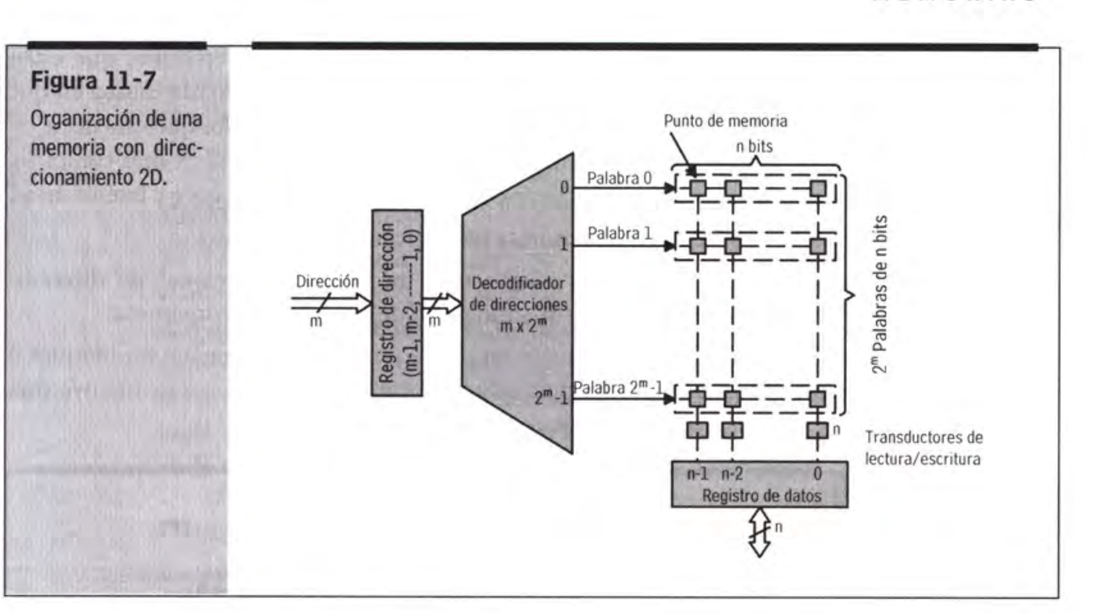


Figura 11-7
Organización de una memoria con direccionamiento 2D.

Direccionamiento 3D

Como se refleja en la figura 11-8, se establecen **n** planos de memoria (uno para cada bit de la palabra). Dentro de cada plano, se selecciona el punto de memoria haciendo coincidir las líneas de selección X e Y.

La dirección de **m** bits se divide en dos partes m_x y m_y, que se decodifican en dos decodificadores de 2^{mx} y 2^{my} salidas, respectivamente. Las señales de salida 2^{mx} y 2^{my} se usan como coordenadas que seleccionan entre las 2^m posiciones de cada plano de memoria.

Figura 11-8
Organización de una memoria con direccionamiento 3D.

Cada plano tiene una pareja de transductores de lectura y escritura, que están conectados a todos los puntos del plano. La ventaja del método 3D estriba en que dos decodificadores de m_x y m_y entradas son mucho más sencillos que uno de $m_x + m_y = m$ entradas. Como inconveniente, el punto de memoria es más complejo, puesto que debe ser capaz de activarse sólo cuando sus dos líneas **x** e **y** estén activas.

2º) Direccionamiento en memorias dinámicas y de propagación

En estas memorias hay que añadir una información adicional de direccionamiento, que se almacena ocupando parte de los puntos de memoria.

La técnica más empleada consiste en empaquetar la información en bloques o registros a los que se añade una cabecera, que, entre otras cosas, incluye una identificación del bloque o registro. Figura 11-9.

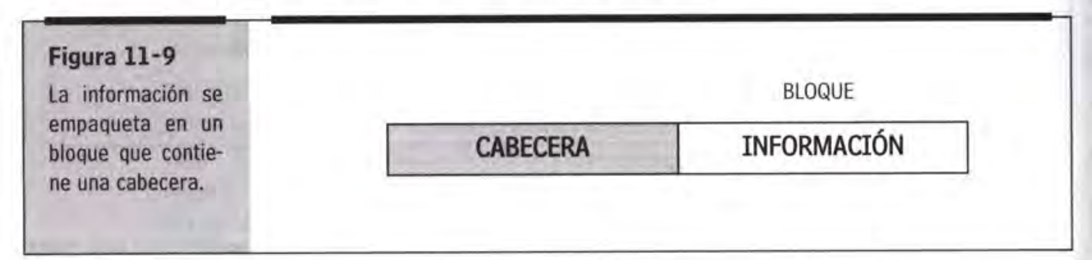


Figura 11-9
La información se empaqueta en un bloque que contiene una cabecera.

La unidad de control del dispositivo deberá interpretar el identificador para poder seleccionar la posición de memoria requerida. Generalmente, estas memorias se usan para seleccionar bloques enteros y no posiciones individualizadas. Esto significa que la lectura y escritura se hacen a nivel de bloque y no de palabra.

Además, dado que el medio o soporte es continuo, es necesario disponer de una señal de reloj que permita diferenciar los puntos de memoria consecutivos. Esta señal de reloj puede ser externa, pero suele grabarse en el medio para evitar problemas de sincronización. Obsérvese que, a la menor diferencia entre el reloj y la velocidad a la que el medio pasa por delante del transductor, se producirá una interpretación errónea de la información, al haberse perdido la sincronización. Si la señal de reloj está grabada, las variaciones de velocidad producen una variación idéntica en el reloj, por lo que no se pierde el sincronismo. Lo mismo ocurre en las memorias de propagación, donde la velocidad de movimiento de la información debe estar sincronizada con el reloj.

Finalmente, se establecen las diferencias entre los dispositivos de **acceso secuencial** frente a los de **acceso directo**. En los primeros sólo existe un transductor, por lo que para acceder a una posición se debe recorrer todo el medio hasta alcanzar la posición deseada. Esto sucede con la cinta magnética.

En los dispositivos de acceso directo hay varios transductores colocados en diversas posiciones, de forma que se puede pasar de una posición a otra seleccionando el transductor adecuado. Dentro de la zona asignada a cada transductor, el funcionamiento sigue siendo de tipo secuencial, pero, al ser menor esta zona,

el tiempo perdido en esperar que la posición deseada alcance el transductor es menor que si todo el dispositivo fuese de acceso secuencial. Ejemplos de memorias dinámicas de acceso directo son los discos y los tambores, y de memoria de propagación de acceso directo, las memorias de burbujas magnéticas.

TIEMPO DE ACCESO

Es el tiempo que tarda una memoria en realizar una operación de lectura. Abreviadamente se llama t_{ACC}, y se mide en ns. Es un parámetro que mide la velocidad de funcionamiento de la memoria, porque determina el tiempo que transcurre entre el instante en que se aplica una nueva dirección a la entrada del dispositivo y la aparición de la información almacenada en la salida. En la tecnología TTL, el tiempo de acceso suele ser menor de 100 ns, siendo algo superior en la tecnología MOS.

DURACIÓN DE LA INFORMACIÓN

En relación con la permanencia en la memoria de la información grabada, hay cuatro posibilidades:

– *Memorias permanentes*

Son las que contienen siempre la misma información y no pueden borrarse. La información puede haberse grabado en el proceso de fabricación de la memoria o puede haberse efectuado posteriormente en un proceso de grabado destructivo o permanente. Como ejemplos de este tipo de memorias se pueden citar las tarjetas y cintas de papel perforado y las memorias de semiconductores tipo ROM con máscara.

En contraposición a este tipo de memorias de sólo lectura, están las de lectura y escritura, que pueden grabarse cuantas veces se quiera. Como alternativa intermedia están las memorias que, para borrarse, precisan de dispositivos especiales, como las de tipo EPROM.

– *Memorias volátiles*

Precisan estar continuamente alimentadas de energía. Si se corta el suministro, se borra la información que almacenan. En contraposición, están las memorias no volátiles, en las que permanece la información aunque se suprima la alimentación.

– *Memorias de lectura destructiva*

Son memorias cuya lectura implica el borrado de la información. Para que la información no desaparezca, se requiere una escritura, posterior a la lectura, que vuelva a grabar lo que se ha leído. El ejemplo clásico de este tipo de memorias es el de las ferritas. Entre las memorias de lectura no destructiva destacan las de semiconductores, discos y cinta magnética.

– *Memorias con refresco*

La información dura solamente un tiempo determinado. Para que no desapa-

rezca la información, hay que regrabarla de forma periódica, operación que recibe el nombre de **refresco.**

11.3. Memorias de semiconductores

Este tipo de memoria es el que se emplea en la actualidad, con carácter universal, como memoria principal de los computadores. Su comportamiento es similar en todas sus variantes y se basa en las características que se exponen en el siguiente apartado.

11.3.1. Funcionamiento general

Para el funcionamiento de una memoria de semiconductores se precisan tres tipos de señales, además de la alimentación.

1ª. Señales de direccionamiento

A este conjunto de señales se le denomina en la tecnología de computadores "autobuses de direcciones" o, más abreviadamente, **bus de direcciones.**

Con las líneas de direcciones se especifica la dirección de la posición de memoria a la que se desea acceder. El número de líneas (**n**) que conforman el bus de direcciones determina la cantidad de posiciones de memoria (2^n).

2ª. Señales de datos

Se les llama **bus de datos** y pueden introducir información desde el exterior, o bien, sacar información al exterior.

3ª. Señales de control

Forman el **bus de control** y están dedicadas a indicar a la memoria la operación que debe realizar. Las más comunes son las tres siguientes:

a) Lectura/$\overline{\text{Escritura}}$ (R/$\overline{W}$)

Ordena a la memoria si se debe leer o escribir la posición de la memoria direccionada.

b) Permiso (E : Enable)

Cuando esta señal está activa, la memoria realiza normalmente la operación de lectura o escritura, pero si está desactivada, las líneas de datos quedan en estado flotante o triestado y no responde a las señales de direcciones y control.

c) Selección de chip (CS)

Su activación permite el funcionamiento de la memoria, mientras que su desactivación lo prohíbe.

En la figura 11-10 se muestra la estructura interna de una memoria de semiconductores de ocho posiciones de 8 bits cada una, con todas las señales necesarias para su correcto funcionamiento.

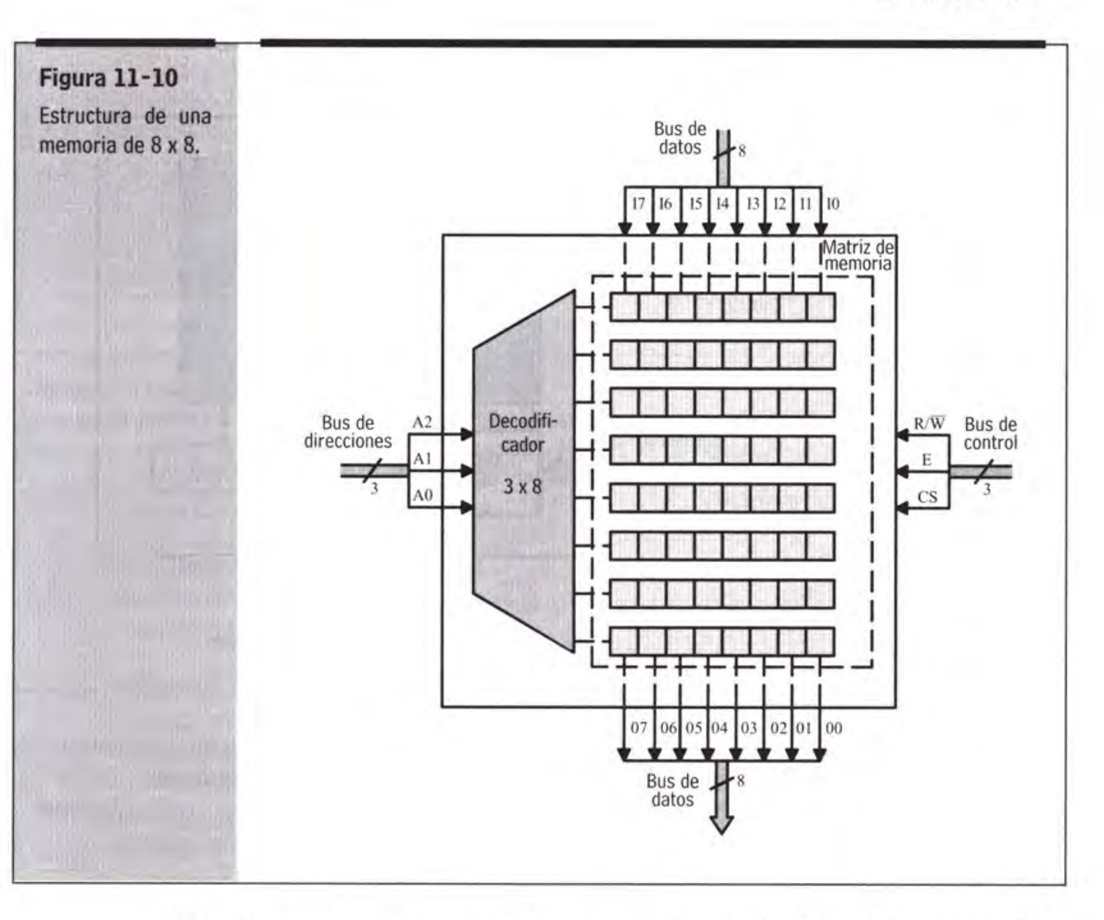


Figura 11-10
Estructura de una memoria de 8 x 8.

El bus de direcciones correspondiente a una memoria de ocho posiciones debe constar de tres líneas (A0, A1 y A2), mientras que el bus de datos adecuado a la memoria de la figura 11-10 deberá disponer de ocho líneas de entrada (I0-I7) para introducir la información a escribir y ocho líneas de salida (O0-O7) para soportar la información en las operaciones de lectura. Finalmente, también se reflejan en dicha figura las tres señales de control típicas:

Lectura/$\overline{\text{Escritura}}$ (R/$\overline{W}$), Permiso (E) y Selección de Chip (CS).

Como en cada instante sólo se puede efectuar una operación de lectura o escritura sobre la posición direccionada, el bus de datos suele ser bidireccional, para que las mismas líneas que actúan como entradas actúen como salidas. De esta forma, la memoria 8 x 8 puede representarse de forma más reducida, como se muestra en la figura 11-11.

A cada una de las ocho posiciones de la memoria descrita corresponde una dirección. En la figura 11-12 se supone que las cuatro primeras posiciones están grabadas con una determinada información. Las otras cuatro posiciones tienen una información no significativa, marcada por una **x**, que representa el valor aleatorio 1 o 0.

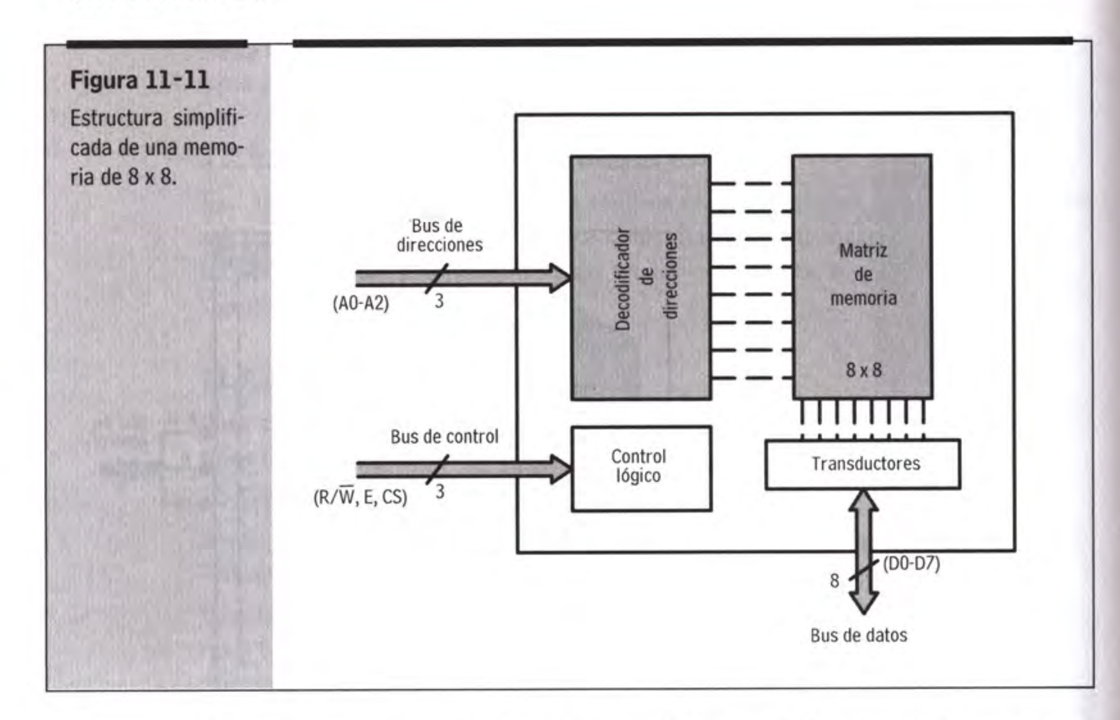


Figura 11-11
Estructura simplificada de una memoria de 8 x 8.

Las fases que componen una operación de lectura de la posición de memoria 2, correspondiente a la estructura de la figura 11-12, son las siguientes:

1ª) Se coloca sobre el bus de direcciones la información A0-A1-A2 = 0-1-0.

2ª) Se activa la señal de lectura $R/\overline{W} = 1$.

3ª) Se da permiso para el funcionamiento de la memoria, E = 1 y CS = 1.

4ª) Transcurrido el tiempo de acceso, aparece en el bus de datos la información contenida en la posición 2:

D7 - D0 = 1111 1000

Figura 11-12
Las cuatro primeras posiciones de la memoria almacenan información válida y las cuatro últimas contienen una información aleatoria, sin significado.

DIRECCIÓN			CONTENIDO								
A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	1	1	1	0	0	1	1	POSICIÓN 0
0	0	1	1	0	0	1	0	0	0	1	POSICIÓN 1
0	1	0	1	1	1	1	1	0	0	0	POSICIÓN 2
0	1	1	0	1	0	1	0	1	0	1	POSICIÓN 3
1	0	0	X	X	X	X	X	X	X	X	POSICIÓN 4
1	0	1	X	X	X	X	X	X	X	X	POSICIÓN 5
1	1	0	X	X	X	X	X	X	X	X	POSICIÓN 6
1	1	1	X	X	X	X	X	X	X	X	POSICIÓN 7

En el caso de querer escribir la información 0000 0000 en la posición de memoria 5, las fases de la operación serían las siguientes:

1ª) Se coloca sobre el bus de direcciones la información A2-A1-A0 = 1-0-1.

2ª) Se activa la señal de escritura $R/\overline{W} = 0$.

3ª) Se da permiso de funcionamiento, E = 1 y CS = 1.

4ª) Se deposita la información a escribir sobre el bus de datos (0000 0000) y transcurrido el tiempo de acceso en escritura, se graba dicha información en la posición 5.

Ejemplo 11-4

A) Indicar los niveles lógicos presentes en las líneas de los buses de direcciones, datos y control correspondientes a la memoria de 8 x 8 de tamaño, descrita en las figuras 11-11 y 11-12, para realizar la operación de escritura del dato 1111 0000 en la posición 7.

SOLUCIÓN

A2-A1-A0 = 1-1-1
$R/\overline{W} = 0$
E = 1 y CS = 1
D7 - D0 = 1111 0000.

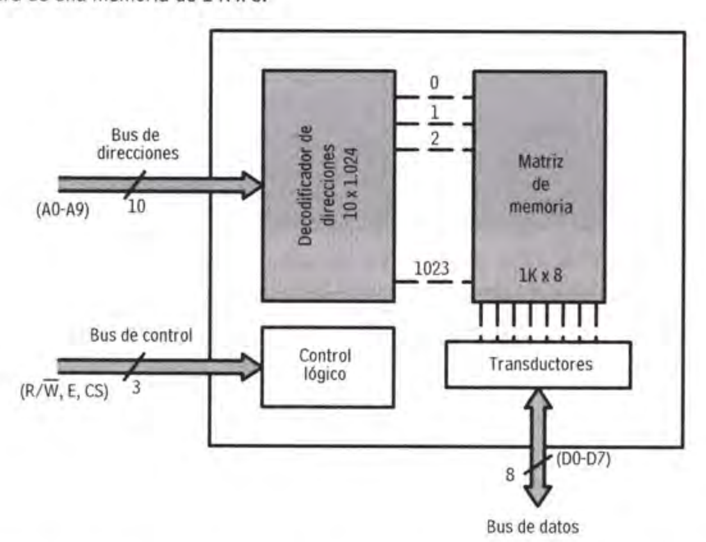
B) Dibujar la estructura interna y la configuración de los buses para definir una memoria de 1K x 8.

SOLUCIÓN

Véase la figura 11-13.

Figura 11-13

Estructura de una memoria de 1 K x 8.



C) Deducir la capacidad de una memoria que consta de 16 líneas (A0-A15) en el bus de direcciones y 16 líneas (D0-D15) en el bus de datos.

SOLUCIÓN

64K x 16

La patita correspondiente a la señal de **selección de chip** sirve para situar el dispositivo de memoria en un rango de direcciones concreto, dentro del área total que puede manejar un computador y que recibe el nombre de **mapa de memoria.** Se analiza este concepto con el siguiente ejemplo.

Ejemplo 11-5

Una UCP o CPU dispone de un bus de direcciones de 16 líneas (A0-A15), un bus de datos de ocho líneas (D0-D7) y un bus de control de tres (CS, R/W y E). Diseñar la lógica necesaria para situar una memoria de lectura y escritura de 2K x 8 de tamaño al principio del mapa de memoria, o sea, ocupando las direcciones comprendidas entre la 0000 H y la 07FF H.

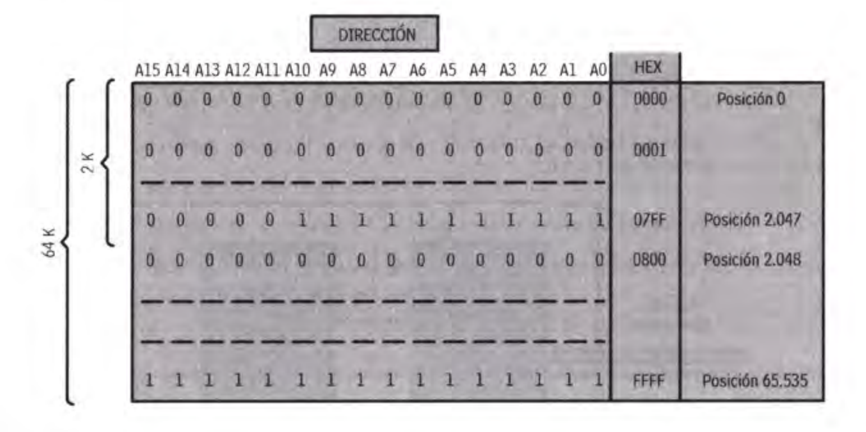
SOLUCIÓN

Como la CPU dispone de 16 líneas de direccionamiento, puede manejar un total de $2^{16} = 64K$ posiciones. Como el bus de datos es de ocho líneas, cada posición consta de 8 bits, lo que supone un mapa de memoria de 64K posiciones de 8 bits cada una.

En la figura 11-14 se ofrece el esquema simplificado del mapa de memoria.

Figura 11-14

Mapa de memoria de la CPU con 16 líneas en el bus de direcciones y situación de las 2K primeras posiciones.



Un dispositivo de memoria de 2K posiciones dispone de 11 líneas de direccionamiento ($2^{11} = 2.048$). Dichas líneas estarán conectadas a las 11 líneas de menos peso del bus de direcciones de la CPU (A0-A10). Como se aprecia en la figura 11-14, las cinco líneas de más peso, A11-A15, deberán valer 0 para que las 2.048 posiciones de la memoria respondan al rango de direcciones comprendido entre la 0000 H y la 07FF H. Esto significa que la señal CS, de permiso de funcionamiento del chip de memoria, deberá activarse cuando se cumpla que A15 = A14 = A13 = A12 = A11 = 0. La figura 11-15 muestra el esquema lógico necesario para que CS se active cuando las cinco líneas de más peso del bus de direcciones de la CPU valgan 0.

Figura 11-15
Una puerta NOR produce una salida de nivel lógico 1 cuando todas sus entradas tienen nivel lógico 0.

El diagrama de adaptación del dispositivo de memoria de 2K x 8 a los buses de la CPU se muestra en la figura 11-16.

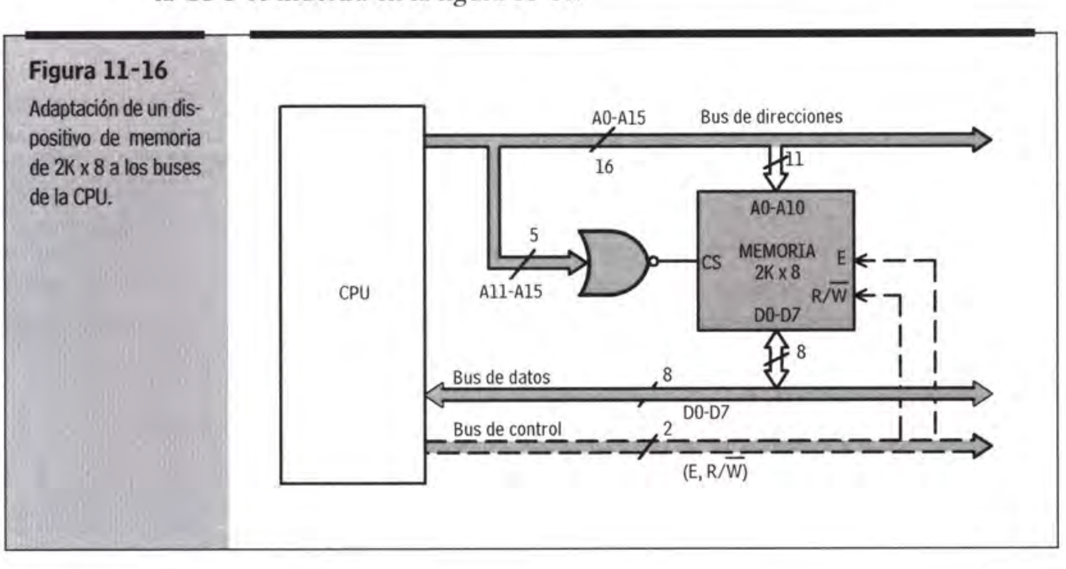


Figura 11-16
Adaptación de un dispositivo de memoria de 2K x 8 a los buses de la CPU.

11.3.2. Clasificación

Las memorias con semiconductores son intrínsecamente memorias de **acceso aleatorio**, que significa que se puede acceder a cualquier posición de la memoria directamente. Este hecho es así porque, para acceder a una posición, basta colocar su dirección en el bus correspondiente. No se precisa pasar previamente por otras posiciones contiguas, como sucede en las memorias de acceso secuencial (discos y cintas magnéticas).

Para clasificar las memorias de semiconductores se utiliza una terminología que resulta un poco confusa, pues usa términos empleados con otros sentidos. Básicamente existen dos tipos fundamentales.

A) ROM: de sólo lectura.

En este tipo de memoria de semiconductores sólo se pueden leer los datos almacenados previamente, bien durante su fabricación o bien en un proceso independiente. Los datos grabados permanecen invariables de forma permanente. Son memorias no volátiles porque, aunque se les suprima la alimentación eléctrica, mantienen inalterada la información grabada.

Las celdas básicas de una memoria ROM son muy sencillas comparadas con las del otro tipo. A veces están construidas con un simple diodo o transistor.

Hay cuatro tipos de ROM.

1ª. ROM o ROM con máscara.

Se trata de la ROM básica caracterizada, porque ha sido el propio fabricante del circuito integrado el encargado de grabar la información que almacena.

2ª. PROM o ROM programable.

Se trata de una memoria de sólo lectura, pero que puede grabar su contenido el propio usuario empleando algunas herramientas (computador personal y grabador de memorias).

3ª. EPROM o ROM programable y borrable.

Es una ROM programable por el usuario, que tiene la propiedad de poder borrarse sometiéndola a una exposición de rayos ultravioleta y volver a grabar una información diferente.

4ª. EEPROM o ROM programable y borrable eléctricamente.

Se trata de una ROM programable y borrable por el usuario, pero con la posibilidad de realizar estas operaciones a nivel individual de las posiciones de la memoria, empleando para el borrado procedimientos eléctricos en lugar de rayos ultravioleta.

B) RAM: de lectura y escritura.

Son memorias de acceso aleatorio cuyas posiciones pueden ser leídas y escritas cuantas veces sea necesario. Su principal inconveniente es que son volátiles, perdiendo la información almacenada en sus celdas cuando se suprime la alimentación. Hay dos tipos de RAM:

1ª. RAM estáticas o SRAM.

Sus celdas consisten en flip-flops que almacenan un bit. Pueden utilizar tecnología bipolar o MOS.

2ª. RAM dinámica o DRAM.

Sus celdas son muy simples y están formadas por un condensador cuya carga representa el bit de información. Tienen el inconveniente de precisar refresco que compense las pérdidas del condensador. Son memorias muy rápidas, de mucha densidad de integración y baratas. Se fabrican con tecnología MOS.

11.4. MEMORIAS ROM

Son memorias de sólo lectura y están diseñadas para mantener la información de forma permanente. Teniendo en cuenta la duración de la información, las memorias ROM son del tipo **no volátil,** pues la falta de alimentación no altera el contenido de sus celdas.

En estas memorias todos los datos quedan grabados durante su fabricación o en una operación independiente. Realizada la grabación, los datos no pueden

modificarse. Se emplea en aquellas aplicaciones que requieren una salida invariable o un programa de trabajo fijo, una y otra vez, como sucede con el programa que tienen almacenado los generadores de caracteres alfanuméricos en sistemas con presentadores visuales y en las tablas de referencia que contienen las calculadoras.

La estructura de las celdas de una ROM es muy sencilla, pues suele estar constituida por un diodo o un transistor. En algunos casos especiales está formada por dos transistores. Esta simplicidad confiere a las ROM unas características muy ventajosas, como sucede con la alta densidad de integración, que permite alcanzar en los CI elevadas capacidades de almacenamiento. También tienen un precio reducido y un tiempo de acceso pequeño. Las ROM bipolares poseen tiempos de acceso inferiores a los 100 ns, mientras que las fabricadas con tecnología MOS son algo más lentas. En la figura 11-17 se muestra la estructura de dos celdas típicas de memoria ROM.

Figura 11-17

En la parte superior, una celda de memoria ROM basada en un transistor y, en la parte inferior, otra celda construida sobre un diodo. Si se efectúa la conexión programable, la celda almacena un bit 1, y, en caso contrario, un bit 0.

La celda formada por un transistor en la figura 11-17 dispone de una conexión en la Base que puede unirse o no con la línea de la señal de selección. Para leer el bit escrito en dicha celda se aplica un voltaje positivo en la señal de selección, que se transfiere a la Base del transistor cuando se ha efectuado la conexión; en esta situación, el transistor se satura, quedando prácticamente $+V_{CC}$ en la resistencia R, haciendo que el bit de información corresponda a un nivel lógico alto, o sea, un 1. Si la conexión entre la Base y la línea de selección está abierta, el

transistor no conduce, dejando sin tensión a la resistencia R, con lo que el bit de información corresponde a un nivel lógico bajo, o sea, un 0.

La celda basada en un diodo en la figura 11-17 tiene un comportamiento similar a la del transistor. En caso de tener realizada la conexión programable, al aplicar + V_{CC} en la línea de selección el diodo queda polarizado directamente y la mayor parte de dicha tensión queda en la resistencia R, lo que representa que el bit de información es un 1. Si la conexión está abierta, al no poder circular corriente por el diodo, el nivel lógico que representa la celda es bajo, es decir, almacena un 0.

En la figura 11-18 se muestra la estructura interna simplificada de una ROM de 4 x 2, a base de celdas con transistores.

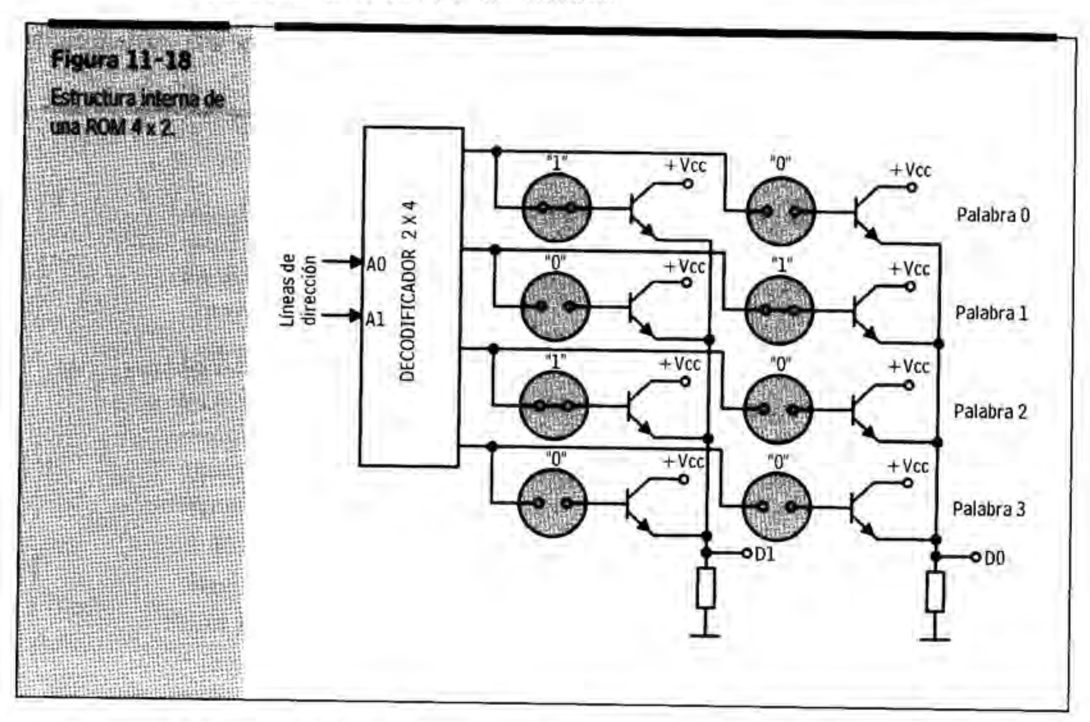


Figura 11-18
Estructura interna de una ROM 4 x 2.

La memoria ROM, representada en la figura 11-18, está formada por cuatro palabras, cada una de dos celdas construidas con un transistor al que se puede programar el conexionado de su Base. Inicialmente, mediante las líneas de dirección A0 y A1 se selecciona cuál de las cuatro palabras se desea leer. Cada palabra consta de 2 celdas que almacenan un 1 si tienen unida la Base del transistor con la línea de selección de la palabra, que procede del decodificador. Si la conexión está abierta, almacenan un 0. En el caso de la ROM del ejemplo, su contenido se muestra en la tabla 11-2.

Tabla 11-2
Direcciones y contenidos de la ROM.

DIRECCIÓN		CONTENIDO	
A0	A1	D0	D1
0	0	0	1
1	0	1	0
0	1	0	1
1	1	0	0

Grabar un bit en una celda de una ROM consiste simplemente en efectuar o no la conexión programable. En el primer caso se escribe un 1 y en el segundo, un 0.

A veces se puede acceder individualmente a las celdas, adoptando la estructura de la memoria una organización matricial, compuesta por filas y columnas. Así, una memoria de 64 bits puede responder a una arquitectura como la presentada en la figura 11-19. Cada celda puede ser accedida individualmente, para lo cual sólo hay que especificar la dirección de la fila y la de la columna a la que pertenece.

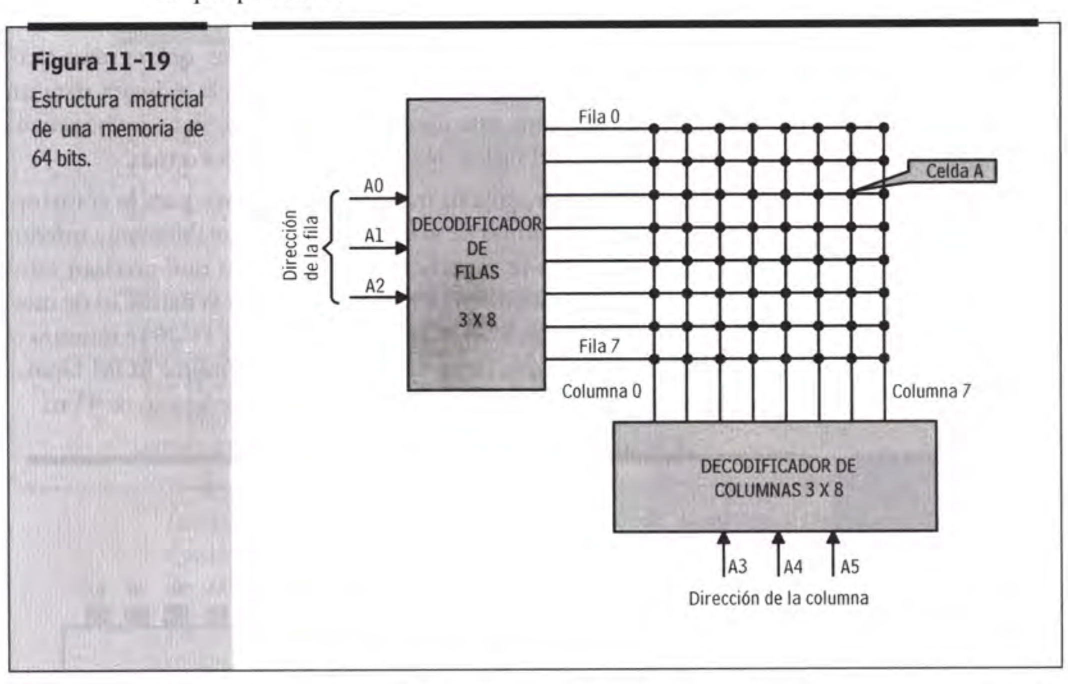


Figura 11-19
Estructura matricial de una memoria de 64 bits.

Para acceder a la celda **A** de la estructura de la memoria correspondiente a la figura 11-19, hay que proporcionar a los respectivos decodificadores la dirección de la fila y la de la columna. Con relación a dicha celda, se debe introducir al decodificador de filas el código de la fila 2 (A0 = 0, A1 = 1 y A2 = 0), y al decodificador de columnas el de la columna 6 (A3 = 0, A4 = 1 y A5 = 1).

Ejemplo 11-6

A) Si en la memoria representada en la figura 11-19 se aplica la dirección A0 = A1 = A2 = A3 = A4 = A5 = 0, deducir cuál es la celda a la que se accede.

SOLUCIÓN

Se accede a la celda situada en la fila 0 y la columna 0.

B) Se dispone de una memoria de 1 Kb (1.024 bits), con estructura matricial y acceso independiente para cada celda. Deducir el número de líneas necesario para determinar la fila y la columna de cada celda.

SOLUCIÓN

Con $1.024 = 2^{10}$ celdas se precisa un total de 10 líneas de direccionamiento, de las cuales cinco estarán destinadas a seleccionar la fila y otras cinco la columna.

La matriz de la memoria se compondrá de $2^5 = 32$ filas y otras tantas columnas (32 x 32 = 1.024).

11.4.1. ROM con máscara

En este tipo de memorias la grabación de los datos se lleva a cabo durante el proceso de fabricación del CI. Son imborrables y el usuario no puede modificar su contenido.

El proceso de grabación está basado en el diseño de una **máscara** sobre un foto-lito. En dicha máscara se determinan las conexiones programables que se van a efectuar y las que van a quedar abiertas. Como la confección de la máscara para un contenido concreto es bastante cara, este tipo de memoria sólo tiene interés cuando se necesitan grandes series, del orden de las 10.000 unidades o más.

Las ROM con máscara bipolares utilizan transistores bipolares para la construcción de cada celda y tienen un tiempo de acceso muy bajo, notablemente inferior a los 100 ns. Sin embargo, dada la superficie de aislamiento que precisan estos transistores, las celdas ocupan bastante superficie, reduciendo la densidad de integración y, en consecuencia, la capacidad máxima. En la figura 11-20 se muestra el esquema simplificado y el diagrama de conexiones de la memoria ROM bipolar 7488 A, que tiene una capacidad de 32 x 8 bits y un tiempo de acceso de 45 ns.

Figura 11-20

Esquema y diagrama de conexionado de la memoria ROM bipolar 7488A, que tiene una capacidad de 32 x 8 bits.

Ejemplo 11-7

Se desea implementar sobre una memoria ROM una tabla fija de valores, que resuelva la siguiente ecuación:

$$Y = (A + B + C + D + E) \cdot 2$$

Las cinco variables de la ecuación (A, B, C, D y E) son de tipo binario y con ellas se pueden realizar 32 combinaciones diferentes. Dichas combinaciones se comportarán como las direcciones de la memoria ROM. En cada dirección habrá grabado un contenido que se corresponde con el valor de la ecuación Y para dichos valores de las variables que forman la dirección. Así, al aplicar a las líneas de direcciones de la memoria (A0 - A4) los valores de las cinco variables binarias de la ecuación, se accede a una posición que guarda el valor del resultado Y.

Se pide:

1º) Confeccionar una tabla con el contenido y la dirección correspondientes a las seis primeras posiciones de la ROM.

2º) Si se introduce a la ROM, como valor de las cinco variables binarias, un 1, indicar a qué dirección de la memoria se accede y cuál será su contenido.

SOLUCIÓN

1º) Véase la siguiente tabla (11-3).

Tabla 11-3

DIRECCIÓN					CONTENIDO $Y = (A + B + C + D + E) \cdot 2$							
A	B	C	D	E								
A0	A1	A2	A3	A4	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	1	0	0	0	0	0

2º) Si A = B = C = D = E = 1, las líneas de direccionamiento valen 1, o sea, A0 = A1 = A2 = A3 = A4 = 1, con lo cual se selecciona la última posición de la memoria, que es la 31 ($11111_{(2}$).

El contenido de la posición 31 será:

$Y = (A + B + C + D + E) \cdot 2 = (1 + 1 + 1 + 1 + 1) \cdot 2$

$Y = 10_{(10} = 0000\ 1010_{(2}$ de donde se deduce que el dato que habrá grabado en la posición 31 será:

D0 = 0, D1 = 1, D2 = 0, D3 = 1, D4 = D5 = D6 = D7 = 0

En la figura 11-21 se ofrece un cuadro de características técnicas de funcionamiento y el diagrama de conexiones de una ROM bipolar, modelo 74187 N, de 1.024 bits organizados en 2.556 palabras de 4 bits cada una.

Las memorias ROM fabricadas con tecnología MOS tienen unas celdas más pequeñas y más lentas que las realizadas con tecnología bipolar. Los CI de memoria ROM, tipo MOS, modelos R2316A y R2316B, fabricados por **Rockwell**, están organizados en 2.048 palabras de 8 bits y tienen un tiempo de acceso comprendido entre 450 y 500 ns. Sus entradas y salidas trabajan con niveles TTL, con una inmunidad al ruido de 0,4 V y un voltaje de alimentación de + 5 V. Funcionan de forma asíncrona y no precisan señales de reloj. En la figura 11-22 se muestra el diagrama de conexionado de estas memorias, en el que se aprecia la presencia de tres patitas destinadas a la selección de chip (**CS1**, **CS2** y **CS3**).

En la figura 11-23 se ofrece el diagrama por bloques de la memoria R2316, compuesto por el decodificador de direcciones (de filas y de columnas), la matriz de celdas y el buffer amplificador de salida, que sólo habilita las salidas cuando se recibe el código correcto por las señales CS1, CS2 y CS3.

El gráfico de la figura 11-24 relaciona el tiempo de acceso t_{ACC}, expresado en ns, con el voltaje de alimentación $\mathbf{V_{CC}}$.

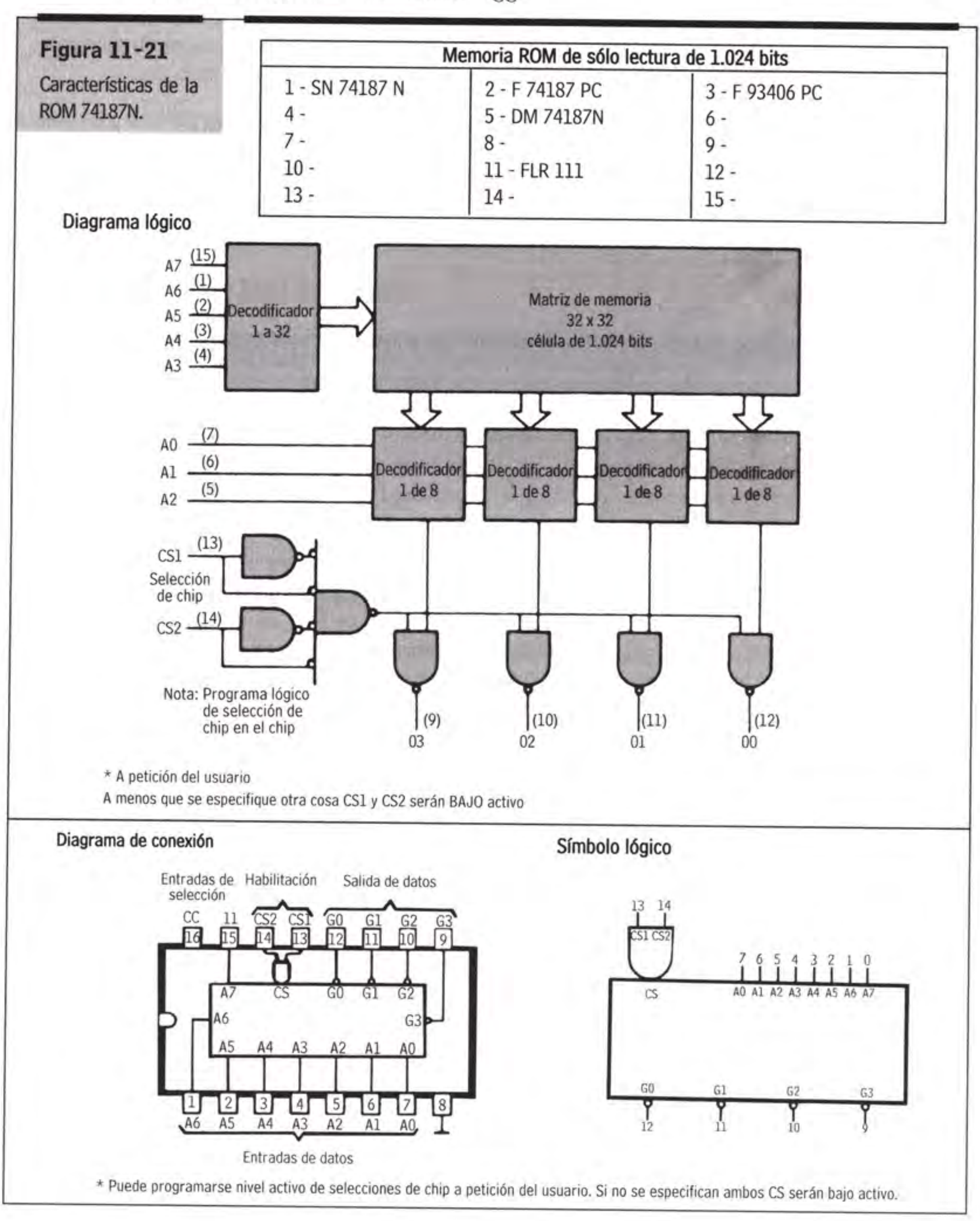


Figura 11-21
Características de la ROM 74187N.

Memoria ROM de sólo lectura de 1.024 bits		
1 - SN 74187 N	2 - F 74187 PC	3 - F 93406 PC
4 -	5 - DM 74187N	6 -
7 -	8 -	9 -
10 -	11 - FLR 111	12 -
13 -	14 -	15 -

Figura 11-21 (Cont.)
Características de la ROM 74187N.

DESCRIPCIÓN.- El dispositivo es una memoria bipolar de sólo lectura de 1.024 bits organizada en 256 palabras de 4 bits. Se utiliza una dirección binaria de cuatro bits para seleccionar la palabra deseada. Las cuatro salidas son colectores libres que permiten uniones de las salidas, para ampliación de memoria en la dirección de palabras. El usuario puede especificar el nivel activo de la puerta de selección de chip de dos entradas.

CS1 y CS2 serán ambas BAJO activo, a menos que se especifique otra cosa por el usuario. La característica de habilitación programable permite la ampliación de memoria a 1.024 palabras sin ninguna puerta externa.

Los contenidos de la memoria quedan (mask programmed) programados según especificaciones del usuario. El usuario puede especificar el código ROM deseado, bien en la forma(s) de Codificación (Coding Form) del dispositivo o mediante tarjetas perforadas utilizando el Formato de Tarjeta de información (Data Card Format).

CONDICIONES DE FUNCIONAMIENTO RECOMENDADAS

PARÁMETRO	MÍN.	TÍP.	MÁX.	UNIDADES
Tensión de alimentación Vcc	4,75	5,0	5,25	Volts.
Margen de temperatura ambiente	0	25	75	°C

CARACTERÍSTICAS ELÉCTRICAS (T_A = 0 °C a + 75 °C, Vcc = ´5,0 V ± 5%

SÍMBOLO	CARACTERÍSTICAS	MÍN.	TÍP.	MÁX.	UNIDADES	CONDICIONES DE PRUEBA (1)
I_{CEX}	Corriente de fuga de salida			40	μA	V_{CC} = 5,25 V, V_{CEX} = 5,25 V Direccionar cualquier salida ALTA
V_{OL}	Tensión BAJA de salida			0,45	Volts	V_{CC} = 4,75 V, I_{OL} = 15 mA Direccionar cualquier salida BAJA.
V_{IH}	Tensión ALTA de entrada	2,0			Volts	Tensión ALTA de entrada garantizada para todas las entradas.
V_{IL}	Tensión BAJA de entrada			*0,85	Volts	Tensión BAJA de entrada garantizada para todas las entradas.
I_F	Corriente BAJA de entrada				mA	V_{CC} = 5,25 V, V_F = 0,45 V
	I_{FA} (Entrada de dirección)			0,8		
	I_{FCS} (Entrada de selección de chip)			0,8		
I_R	Corriente ALTA de entrada				μA	V_{CC} = 5,25 V, VR = 4,5 V
	I_{RA} (Entradas de dirección)			40		
	I_{RCS} (Entradas de selección de chip)			40		
I_{CC}	Corriente de alimentación		114	130	mA	V_{CC} = 5,25 V salidas abiertas a masa y chip seleccionado.
C_0	Capacidad de salida		6,5		P^F	V_{CC} = 5,0 V, V_0 = 5,0 V, f = 1,0 MHz
V_{CD}	Tensión en diodo limitador de ent.		- 0,8	- 1,0	Volts	V_{CC} = 4,75 V, I_A = - 5,0 mA

Figura 11-22
Diagrama del conexionado de patitas de las memorias ROM R2316A y B.

Figura 11-23
Estructura interna de la memoria ROM, tipo MOS, R2316.

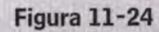
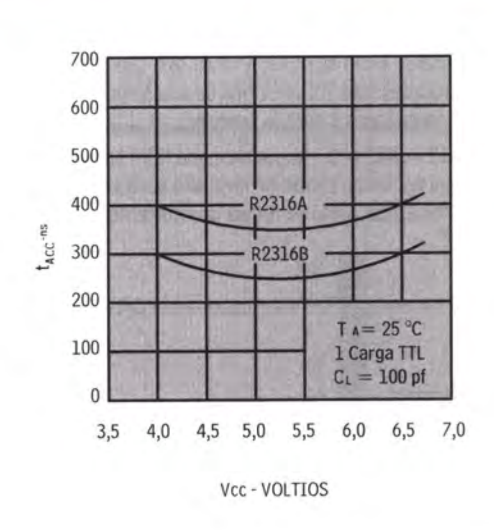


Figura 11-24

Gráfico que relaciona el tiempo de acceso en ns con el voltaje de alimentación de las memorias R2316.



Ejemplo 11-8

Se desea conectar una memoria R2316 de 2K x 8 a una CPU, que dispone de un bus de direcciones de 14 líneas y un bus de datos de ocho. Diseñar el esquema lógico que emplaza dicha memoria en las últimas direcciones del mapa de memoria de la CPU, cuyo tamaño es de $2^{14} = 16K$ posiciones de 8 bits cada una.

SOLUCIÓN

El mapa de memoria ocupa las siguientes direcciones distribuidas según la siguiente tabla (11-4).

Tabla 11-4

	POSICIÓN	DIRECCIÓN													
		A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
16K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	14.335	1	1	0	1	1	1	1	1	1	1	1	1	1	1
2K	14.336	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	16.383	1	1	1	1	1	1	1	1	1	1	1	1	1	1

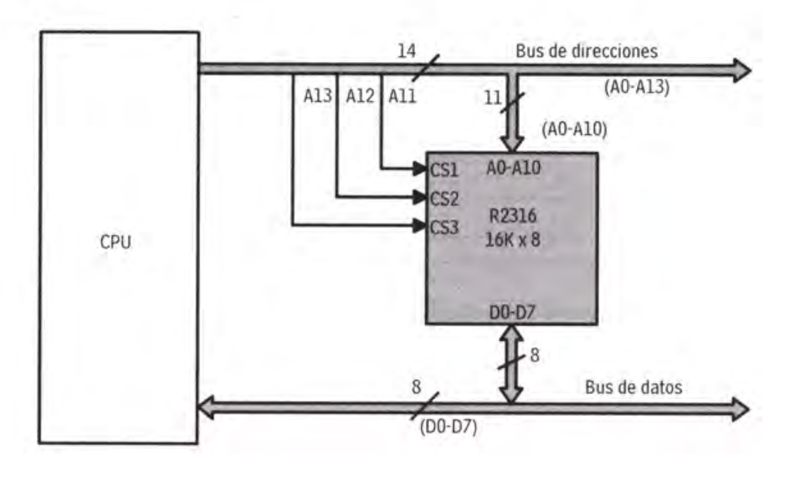
Ejemplo 11-8 (Cont.)

Si se desea que el CI R2316 ocupe las últimas posiciones del mapa de memoria, funcionará con el rango de direcciones comprendido entre la 14.336 y la 16.383, que en binario se caracteriza porque los 3 bits de más peso de dichas cantidades vale 1. Los 11 bits de menos peso restantes cambian su valor desde todos 0 para la posición inferior hasta todos 1 para la superior.

Dado que A13 = A12 = A11 = 1 y que la memoria ROM empleada dispone de tres líneas de selección de chip (CS1, CS2 y CS3) que necesitan recibir un nivel alto para permitir el funcionamiento del CI, se conectan estas señales con las primeras, tal como se refleja gráficamente en el esquema de la figura 11-25.

Figura 11-25

Conexionado de la memoria R2316 a los buses de la CPU para ocupar las 2K posiciones últimas del mapa de memoria.



11.4.2. PROM

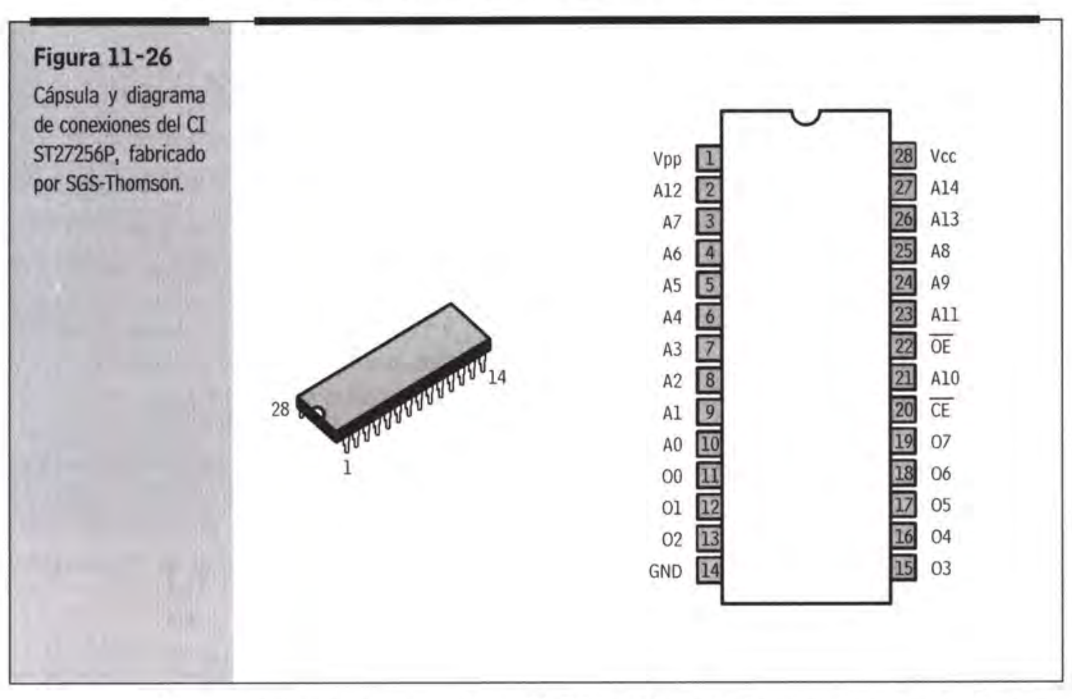
A estas memorias también se les llama **programables una vez** *(OTP: One Time Programmable)*, porque su contenido puede definirlo el usuario una sola vez. Si hay que modificar ese contenido, la memoria grabada no tiene utilidad y hay que programar otra nueva.

Así como en las memorias "ROM con máscara" se definía sobre el negativo de la máscara la posible conexión de la Base del transistor en las celdas que debían almacenar un bit 1, en las celdas vírgenes de las PROM la Base del transistor viene conectada de fábrica con un **fusible**. Si se desea que una celda almacene un 1, se deja intacto su fusible, mientras que si se desea que guarde un 0, se funde dicho fusible. Para esta operación se emplea el **quemador de fusibles**, que es una herramienta conectada a un computador personal desde el que se envía la información a grabar en las celdas de la PROM. La ayuda del computador en la programación de este tipo de memorias hace esta labor rápida, sencilla y fácil de comprobar.

Se describen las características más relevantes de una memoria PROM (Programable una sola vez), fabricada por **SGS-Thomson** con tecnología NMOS; se trata del modelo **ST27256P**. En la figura 11-26 se presenta la cápsula que contiene dicha memoria, junto con el diagrama de conexiones.

Figura 11-26

Cápsula y diagrama de conexiones del CI ST27256P, fabricado por SGS-Thomson.



La memoria está organizada en 32.768 palabras de 8 bits y dispone de 15 líneas para el direccionamiento (A0-A14). Se alimenta con + 5 V y tiene un tiempo de acceso de 200 ns. Utiliza las siguientes señales de control.

1ª. $\overline{\text{OE}}$ (Habilitación de salida).

La desactivación de esta señal bloquea la salida de la información de la posición direccionada. En tal situación, las patitas O0-O7 quedan en alta impedancia.

2ª. $\overline{\text{CE}}$ (Habilitación del chip).

Cuando esta señal está desactivada recibiendo nivel lógico alto, la memoria queda en estado de espera, llamado ***standby***, en el que se reduce considerablemente el consumo de energía. La máxima corriente en *standby* es de 40 mA. Dado que la corriente en funcionamiento normal es de 100 mA, supone un ahorro mínimo del 60% de energía.

Teniendo el estado lógico aplicado a las dos señales de control de la memoria **ST27256P**, en la siguiente tabla (11-5) se ofrecen los principales modos de funcionamiento.

Tabla 11-5
Modos de funcionamiento del CI ST27256P.

MODO DE FUNCIONAMIENTO	PATITAS CE	CE	V_{PP}	V_{CC}	SALIDAS 00 - 07
LECTURA (FUNCIONAMIENTO NORMAL)	BAJO	BAJO	V_{CC}	V_{CC}	D_{OUT} SALIDA INFORMACIÓN
SALIDAS INHABILITADAS	BAJO	ALTO	V_{CC}	V_{CC}	ALTA IMPEDANCIA
STANDBY	ALTO	X	V_{CC}	V_{CC}	ALTA IMPEDANCIA
PROGRAMACIÓN	BAJO	ALTO	V_{PP}	V_{CC}	D_{IN} ENTRADA INFORMACIÓN
VERIFICACIÓN	ALTO	BAJO	V_{PP}	V_{CC}	D_{OUT} SALIDA INFORMACIÓN

La tensión V_{CC} de alimentación es de + 5 V con una tolerancia admisible del 5%. La tensión V_{PP} empleada en la fase de programación y verificación es de + 12,5 V. Inicialmente, al recibirse de fábrica, todas las celdas de esta memoria están conteniendo un 1, debiéndose introducir de forma selectiva, en la fase de programación, las posiciones que han de contener un 0. En el proceso de programación se aplica una tensión V_{PP} = + 12,5 V y se aplica la información a cargar en las celdas por las líneas de datos, al mismo tiempo que por las líneas de direcciones se determina la posición a escribir. En esta fase la señal de control debe estar a nivel lógico bajo.

11.4.3. EPROM

Estas memorias reciben esta denominación por su propiedad de poder ser borradas (*erasable*) y reprogramadas las veces que se precise. Para borrar la información grabada previamente en una EPROM, se la somete a la acción de la luz ultravioleta durante un tiempo comprendido entre 10 y 20 minutos. Con esta finalidad existe una ventana de cristal en la superficie de la cápsula que contiene estas memorias. Tras la acción de los rayos ultravioleta, todas las celdas quedan conteniendo un 1. Los rayos ultravioleta ocasionan una corriente electrónica que modifica la carga de los transistores MOS que conforman las celdas, bloqueándoles y originando un nivel lógico alto en sus salidas. Para grabar, posteriormente, un 0 en las celdas, se aplica un elevado voltaje, que suele estar comprendido entre 10 y 25 V, que modifica la carga eléctrica en la región de la puerta del transistor, haciéndole conducir y produciendo un nivel bajo en su salida.

Se describen las características más importantes de la memoria **M27512**, fabricada con tecnología NMOS por **SGS-Thomson**. Se trata de una EPROM, programable eléctricamente y borrable con rayos ultravioletas. Tiene un tiempo de acceso máximo de 200 ns (modelo M27512-2F1) y una alimentación de + 5 V, siendo compatible con niveles TTL. Está organizada en 64K palabras de 8 bits y en la figura 11-27 se muestra el encapsulado, provisto de ventana de cristal, y el diagrama de conexiones.

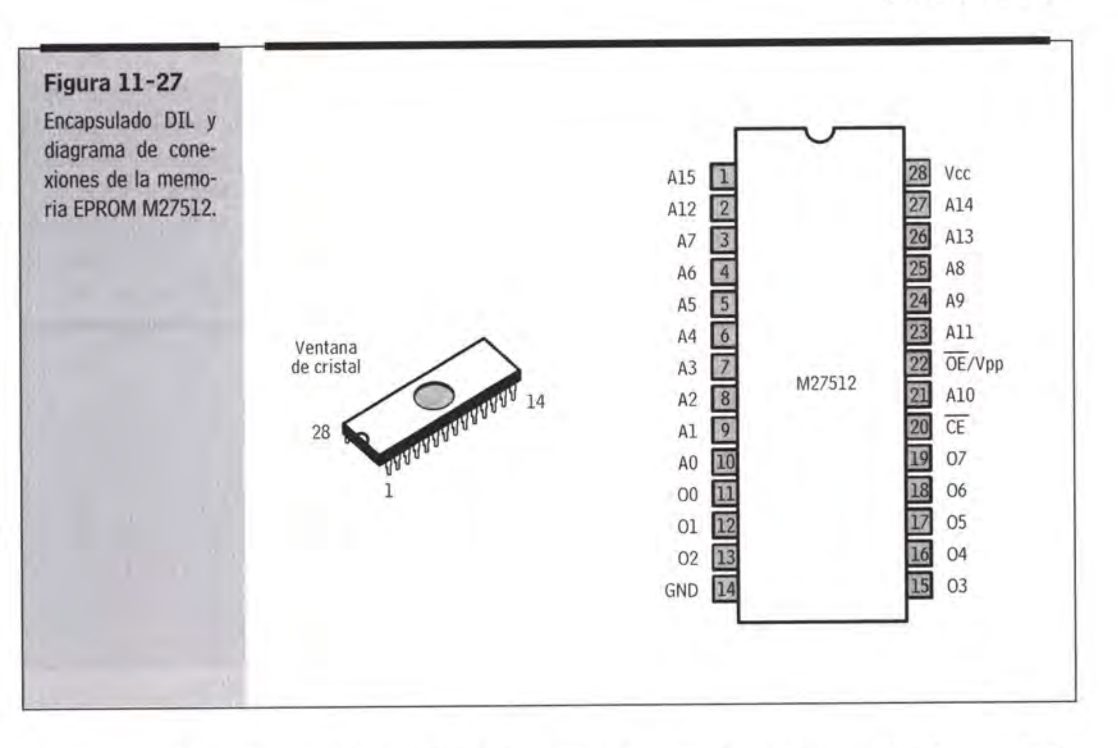


Figura 11-27
Encapsulado DIL y diagrama de conexiones de la memoria EPROM M27512.

Las 28 patitas de la EPROM, agrupadas por funciones, se distribuyen de la siguiente forma:

1) Alimentación: **Vcc y GND (Tierra)**.

2) Líneas de direcciones: **A0-A15**.

3) Salida de datos: **O0-O7**.

4) Señales de control: **$\overline{\text{OE}}$/Vpp y $\overline{\text{CE}}$**.

La señal $\overline{\text{OE}}$/Vpp soporta dos misiones. En primer lugar, cuando se está programando la memoria, por esta patita se introducen los impulsos de 12,5 V de magnitud y 25 ms de duración, por cada byte a grabar, el cual se aplica por las patitas O0-O7, al mismo tiempo que la señal $\overline{\text{CE}}$ = 1. En segundo lugar, cuando la memoria trabaja normalmente, esta patita debe estar conectada a tierra o nivel bajo, lo mismo que $\overline{\text{CE}}$. En el caso que $\overline{\text{CE}}$ = 0 y $\overline{\text{OE}}$ /Vpp = 1, las patitas de salida O0-O7 quedan en estado de alta impedancia.

Aplicando un nivel lógico alto a la patita $\overline{\text{CE}}$, la memoria trabaja en modo *standby* o "mantenimiento", en el que se reduce notablemente el consumo de energía.

Para borrar la información grabada en una EPROM hay que someterla durante 15 o 20 minutos a la acción de luz ultravioleta de una longitud de onda de 4.000 Ångstrom, aproximadamente. La luz generada por las lámparas fluorescentes domésticas tarda en borrar el contenido de estas memorias unos tres años, mientras que la luz solar directa las puede borrar en una semana. Se recomienda tapar la ventana de cristal (una vez grabada la memoria) para evitar la entrada de cualquier tipo de radiación luminosa.

Otro modelo de memoria EPROM, programable eléctricamente y borrable con rayos ultravioleta, es el **μPD 27C1000D**, fabricada por **NEC** con tecnología CMOS. Está organizada en 128K palabras de 8 bits, que suponen un total de 1 M bits de capacidad. Su tiempo de acceso varía entre 150 y 250 ns, según las diversas versiones comercializadas. Consume una corriente máxima de 50 mA, pero puede funcionar en standby con un consumo de 100 μA. Como se puede apreciar en la figura 11-28, la única diferencia con el modelo M27512 descrito anteriormente reside en el número de patitas destinadas al direccionamiento.

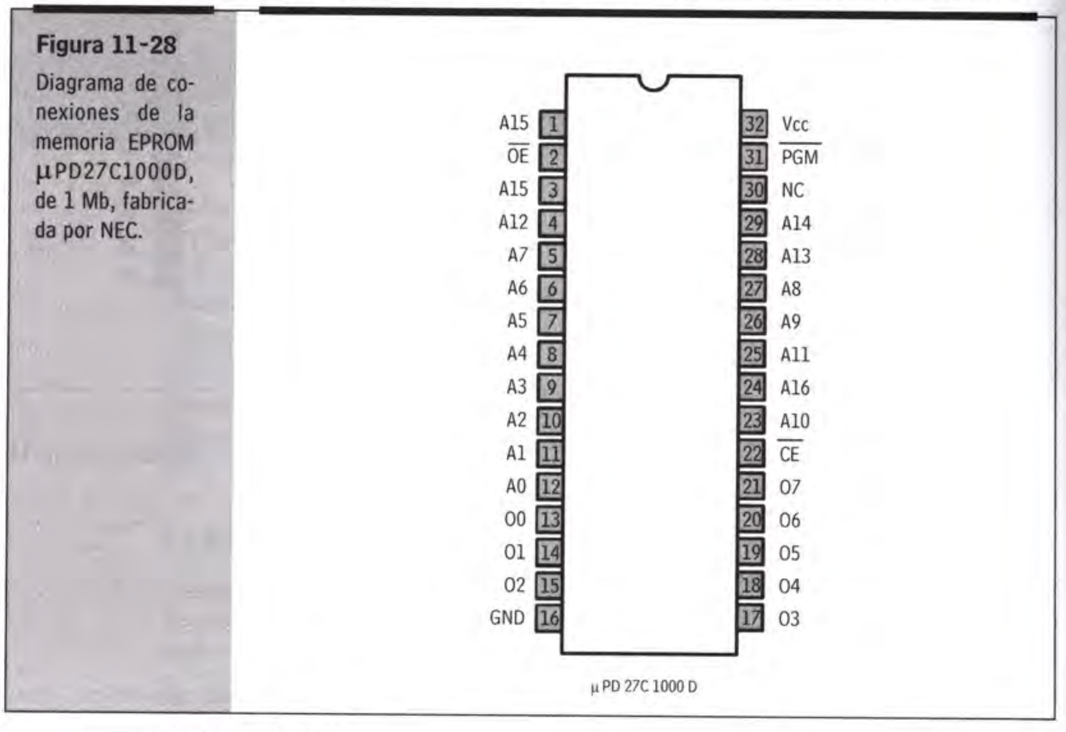


Figura 11-28
Diagrama de conexiones de la memoria EPROM μPD27C1000D, de 1 Mb, fabricada por NEC.

11.4.4. EEPROM

Se trata de un desarrollo tecnológico reciente, que permite grabar y borrar eléctricamente en este tipo de memorias. Como la grabación y el borrado se realizan aplicando elevados voltajes a ciertas regiones de los transistores MOSFET que constituyen las celdas de memoria, ambas operaciones pueden efectuarse con la misma herramienta. Además, los tiempos de borrado son del orden de algunos ms, en comparación con los 15 o 20 minutos que tomaba el borrado con rayos ultravioleta. También la grabación es mucho más rápida. Una ventaja adicional es la posibilidad de poder reprogramar palabras con carácter individual.

Se exponen las características más notorias de la memoria **μPD 28C64/D**, que se trata de una memoria EEPROM organizada en 64K palabras de 8 bits. Se alimenta con + 5 V y el tiempo de acceso es de 250 ns en la versión D-25 y 300 ns en la D-30. En funcionamiento normal consume un máximo de 20 mA y en mantenimiento o *standby* 100 μA. En la figura 11-29 se ofrece el diagrama de conexionado y la estructura interna de este modelo de memoria.

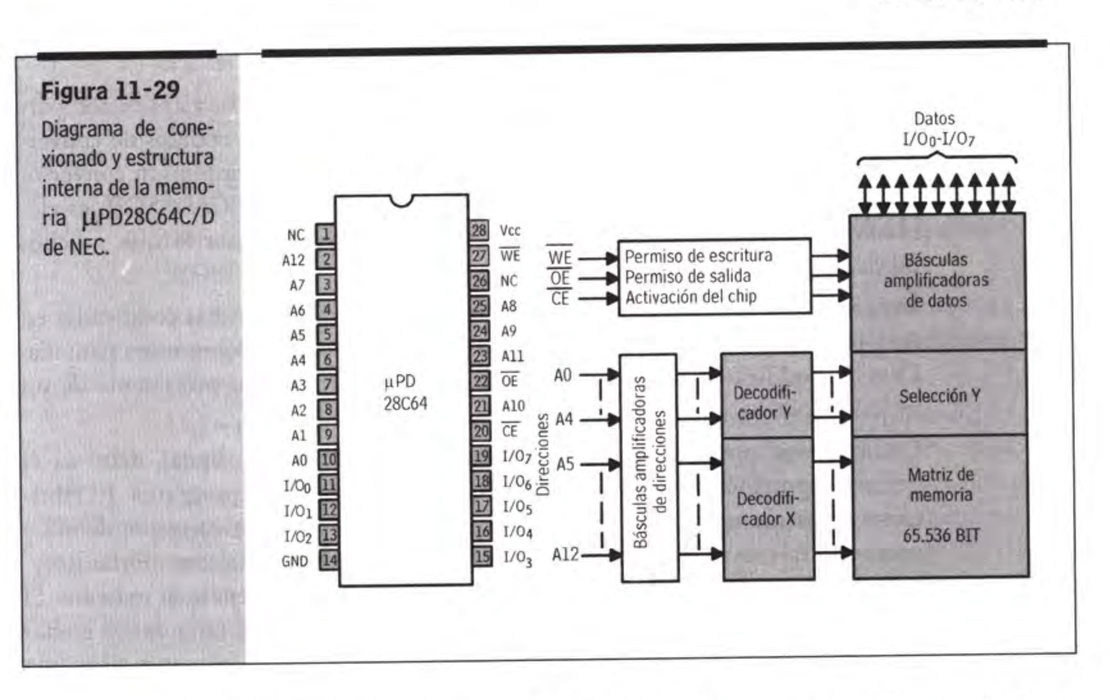


Figura 11-29
Diagrama de conexionado y estructura interna de la memoria µPD28C64C/D de NEC.

Se ofrece a continuación una tabla (11-6) en la que se indican los estados lógicos que deben poseer las tres señales de control para que la memoria **µPD 28C64/D** trabaje en los diversos modos posibles.

Tabla 11-6
Modos de funcionamiento del CI µPD28C64C/D

	PATITAS			
MODO DE FUNCIONAMIENTO	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	I/O_0 - I/O_7
LECTURA	BAJO	BAJO	ALTO	D_{OUT}
STANDBY	ALTO	X	X	ALTA IMPEDANCIA
ESCRITURA	BAJO	ALTO	BAJO	D_{IN}
BORRADO	BAJO	+15 V	BAJO	D_{IN} = ALTO
INHIBICIÓN	X	BAJO	X	———
ESCRITURA	X	X	ALTO	———

Entre los modos de funcionamiento de esta memoria existe uno destinado al borrado de todo su contenido. Las operaciones de autoborrado y programación consumen un tiempo máximo de 10 ms. En cuanto a las formas de escritura, existe una que permite grabar información a cada palabra individualmente (1 byte) y otra que permite escribir una página de 32 bytes, descomponiéndose en este caso la estructura de la memoria en 256 páginas.

11.4.5. Aplicaciones de las memorias ROM

Las ROM que han sido programadas por el fabricante para una aplicación particular son generalmente un producto estándar y contienen códigos de conversión, generadores de caracteres, decodificadores, etc. Por ejemplo, un conversor de código BCD a binario o de código alfanumérico EBCDIC a ASCII, es elemento imprescindible en los computadores y periféricos, por lo que muchos fabricantes los ofrecen programados previamente.

Un generador de caracteres alfanuméricos puede recibir entradas codificadas en ASCII y producir salidas alfanuméricas para unos visualizadores o una pantalla. Otros decodificadores, muy frecuentes, transforman la señal procedente de un teclado en código ASCII.

Cuando es el propio usuario el que diseña una aplicación digital, debe ser él quien proporcione los contenidos específicos que forman el programa. El fabricante de la memoria construye en un solo paso la máscara correspondiente y procede a fabricar los chips encargados. Si es alta la cantidad de memorias necesaria (varios miles), resulta económico que el fabricante prepare la máscara. Si sólo se precisan unos pocos chips, conviene buscar una ROM estándar ya grabada o usar una PROM como alternativa. Además, es muy frecuente usar una PROM para construir el prototipo y realizar las pruebas de puesta a punto antes de encargar la fabricación de las ROM precisas.

Las ROM comunes programadas se usan en los computadores para almacenar microprogramas o para cubrir un objetivo similar en terminales inteligentes y sistemas de control de procesos. En estas aplicaciones funcionan como si fuesen RAM, excepto que su programa no hay que cargarlo, por lo que a veces a este tipo de microprogramación se llama "estática", en oposición a la "dinámica".

Uno de los mejores ejemplos de utilización de una ROM es el almacenamiento de tablas de consulta. Por ejemplo, una ROM se puede usar en una calculadora para almacenar los resultados de una multiplicación de dos números comprendidos entre el 0 y el 9. De esta forma, durante una operación de multiplicación, el circuito calculador puede simplemente introducir el multiplicando y el multiplicador con dos direcciones de la ROM y leer el producto a la salida. Esto supone un considerable ahorro de etapas en comparación con la multiplicación directa de números binarios. La multiplicación de varios dígitos está basada en la repetición de la operación de dos dígitos.

Otra aplicación común de las ROM es la sustitución de circuitos de lógica combinacional. Cada uno de estos circuitos dispone de varias entradas, que determinan el estado de una o más salidas, de acuerdo con la correspondiente tabla de la verdad. Si las direcciones de la ROM representan las entradas al circuito combinacional y las salidas de la memoria contienen el valor de las salidas del circuito, la ROM puede programarse con un duplicado de cada combinación de la tabla de verdad y así simular completamente el circuito combinacional completo. Esto sucede cuando se usa la ROM como conversor de códigos, o como decodificador. Los **PLA** o "conjuntos lógicos programables" son una variante de este concepto aplicado a las ROM.

11.5. Memoria RAM

Las celdas de estas memorias pueden ser leídas y escritas indistintamente. Son de acceso aleatorio, lo que significa que se puede acceder a cualquiera de ellas sin seguir un orden preestablecido. Según el tipo de construcción usado en las celdas, las RAM se clasifican en **estáticas** y **dinámicas**. Si se tiene en cuenta la tecnología de fabricación empleada, puede distinguirse entre las **bipolares** y las **MOS**.

Las RAM reciben el nombre de "memorias vivas", porque su característica primordial es que se pueden leer, escribir y, luego, volver a ser leídas, repitiendo este proceso cuantas veces se desee.

La lectura no es destructiva, o sea, cuando se lee una celda, la información permanece invariable en ella hasta que no se efectúe una operación de escritura. Un grave inconveniente de las RAM es su volatilidad, que supone la pérdida de la información almacenada en las celdas cuando se elimina la alimentación de energía eléctrica. Últimamente se han presentado en el mercado algunas versiones de **RAM no volátiles**, que mantienen la información escrita en sus celdas, pasando al estado de *standby* y empleando unas minúsculas baterías.

La estructura interna de las memorias RAM puede adoptar una de las dos siguientes configuraciones:

1ª) A cada dirección corresponde una celda o bit.

2ª) A cada dirección corresponde una posición con varias celdas, que suelen ser 4, 8 o 16.

En la figura 11-30 se muestra la estructura de una memoria de 256 x 1, configurada en una matriz de 16 filas y 16 columnas.

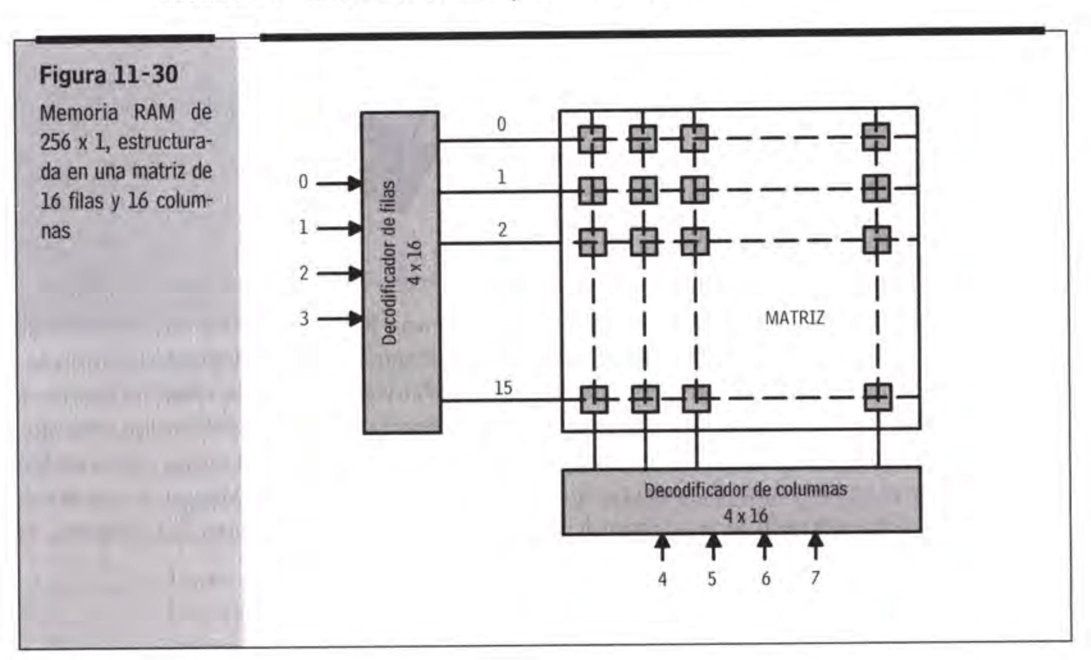


Figura 11-30

Memoria RAM de 256 x 1, estructurada en una matriz de 16 filas y 16 columnas

Mediante ocho líneas de direcciones, cuatro para el decodificador de filas y otras cuatro para el de columnas, se controla la matriz presentada en la figura 11-30. El cruce de la fila y la columna seleccionadas determina la celda a la que se accede.

Si se precisase una memoria RAM de 256 x 4, puede implementarse utilizando cuatro RAM de 256 x 1, tal como se muestra en la figura 11-31, de la que se indica la misión de las diversas señales que aparecen en ella.

A0-A7: líneas de direcciones.

E1-E4: líneas de entrada de datos.

S1-S4: líneas de salida de datos en operaciones de lectura.

$R/\overline{W}$: determina si se realiza una operación de lectura o escritura.

CS: su activación permite el funcionamiento del chip.

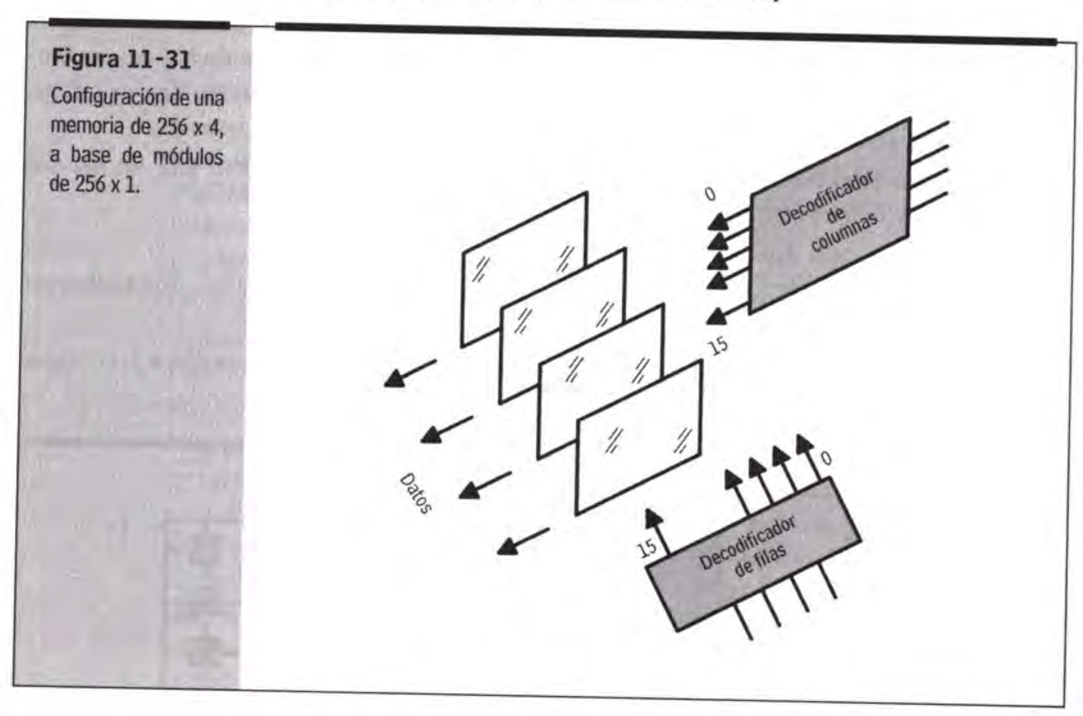


Figura 11-31
Configuración de una memoria de 256 x 4, a base de módulos de 256 x 1.

Para escribir un dato en una posición de una RAM, se coloca en las líneas de direcciones el código de la dirección correspondiente, se deposita la información a grabar en las líneas de entrada de datos y se activa la señal de escritura $R/\overline{W} = 0$, con CS = 1. En las RAM dinámicas la aplicación del dato se debe sincronizar mediante una señal de reloj independiente, mientras que en las estáticas se introduce de forma asíncrona. En una operación de lectura se comienza direccionando la posición a leer, se pone $R/\overline{W} = 1$ con CS = 1, y, después, se recoge la información en las líneas de salida de datos.

11.5.1. Características de las RAM estáticas o SRAM

Las celdas de este tipo de memoria RAM mantienen la información escrita en ellas mientras reciben alimentación eléctrica. Se construyen con transistores bipolares o MOS, siendo este tipo el más empleado por ocupar menos superficie de silicio, entre otras cosas.

En la figura 11-32 se presenta el esquema de una celda bipolar correspondiente a una SRAM. Está basada en dos transistores que configuran un biestable, que representa un bit 1 en el caso de que conduzca T1 y esté bloqueado T2. En esta situación, para leer la celda, se selecciona la línea LW adecuada a través del decodificador de filas, enviando por dicha línea un impulso de tensión positiva de 3 V de amplitud. Este impulso origina la transferencia de la corriente del Emisor conectado a LW al Emisor conectado a LD, línea de datos. La corriente transferida a LD es detectada por un amplificador de lectura aplicado a LD y $\overline{LD}$, que habrá sido preseleccionado con anterioridad mediante el decodificador de columnas. La señal referida se interpreta como la existencia de un nivel lógico 1 grabado en la celda.

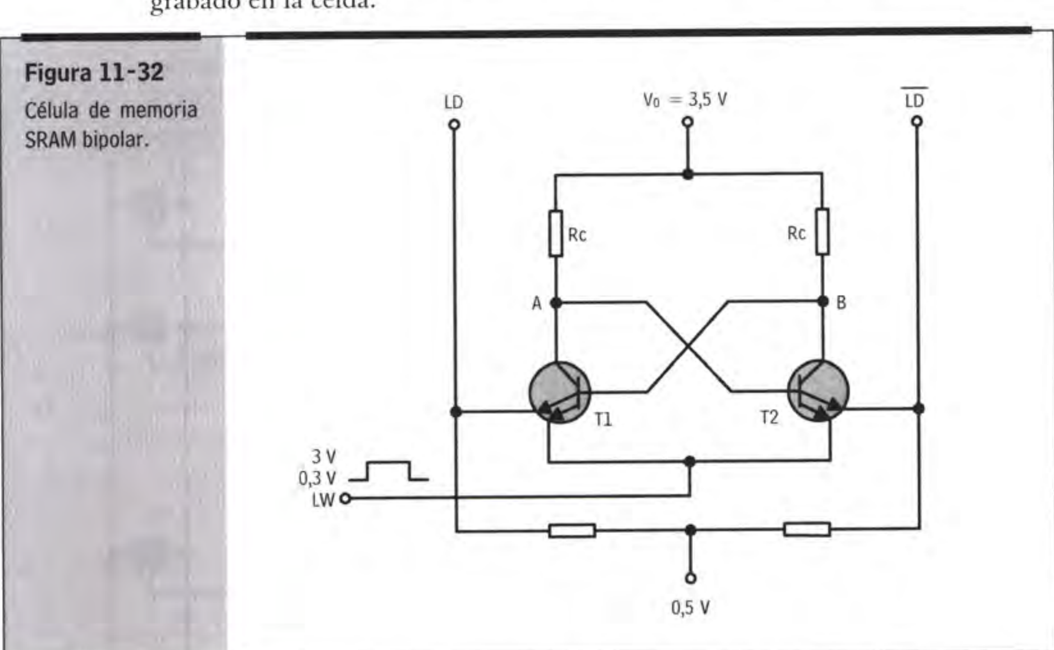


Figura 11-32
Célula de memoria SRAM bipolar.

La estructura interna de otra celda SRAM se ofrece en la figura 11-33, en la que sus transistores son de tipo "uniemisor" y el acoplamiento con las líneas de datos se lleva a cabo con diodos **Schottky** (D1 y D2).

En la figura 11-34 se ofrece el esquema de la estructura matricial de una SRAM bipolar, en la que está seleccionada la celda que ocupa la 2ª fila y la 2ª columna al activar LD2, $\overline{LD2}$ y LW2.

La ventaja de usar tecnología bipolar en la construcción de las SRAM deriva de la gran velocidad que se alcanza y que supone manejar tiempos de acceso inferiores a 30 ns.

Figura 11-33
Célula de SRAM bipolar con transistores uniemisores y diodos Schottky.

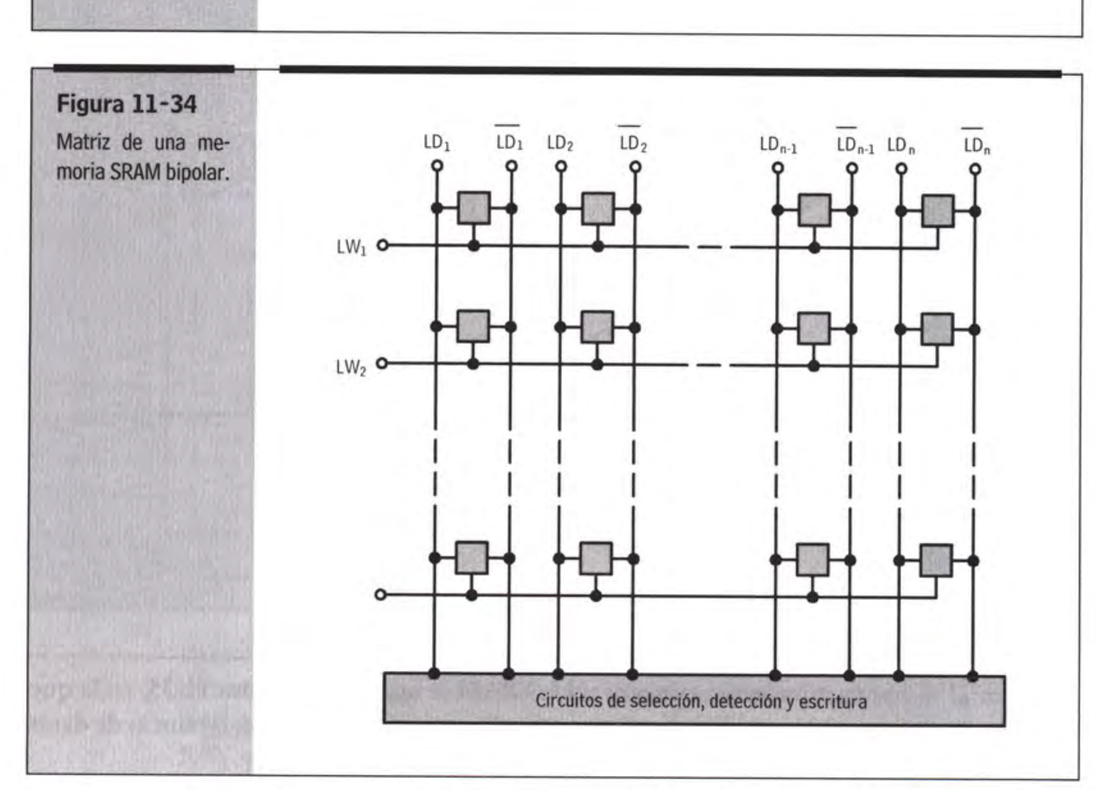


Figura 11-34
Matriz de una memoria SRAM bipolar.

En la práctica se prefiere utilizar las SRAM de tecnología MOS por las siguientes ventajas:

1ª) Mayor densidad de integración, pues cada celda ocupa menos superficie. Esta reducción del área de ocupación de las celdas se debe a la utilización de transistores MOS como resistencias.

2ª) Son más baratas.

3ª) Consumen menos potencia.

En la figura 11-35 se muestra la estructura interna de una memoria RAM, formada por cuatro posiciones con dos flip-flops cada una; mediante las líneas de dirección A0 y A1 y las puertas AND1-AND4 se determina la posición que se desea leer o escribir. Así, por ejemplo, si A0 = 0 y A1 = 1 se selecciona la posición 2, puesto que se abre o activa la AND3. En caso de escritura, la AND5 recibe dos niveles lógicos altos en sus entradas, puesto que CS = 1 y $R/\overline{W} = 0$. Consecuentemente, se aplicará un nivel alto a una de las entradas de las puertas AND6, AND7, AND8 y AND9, obteniéndose un nivel alto en la salida de la AND8, que tiene altas sus dos entradas. La salida de la AND8 actúa como señal de reloj de FF31 y de FF32, cargándose en ellos la información que reciben por sus entradas D, que se aplican por I1 e I2, respectivamente.

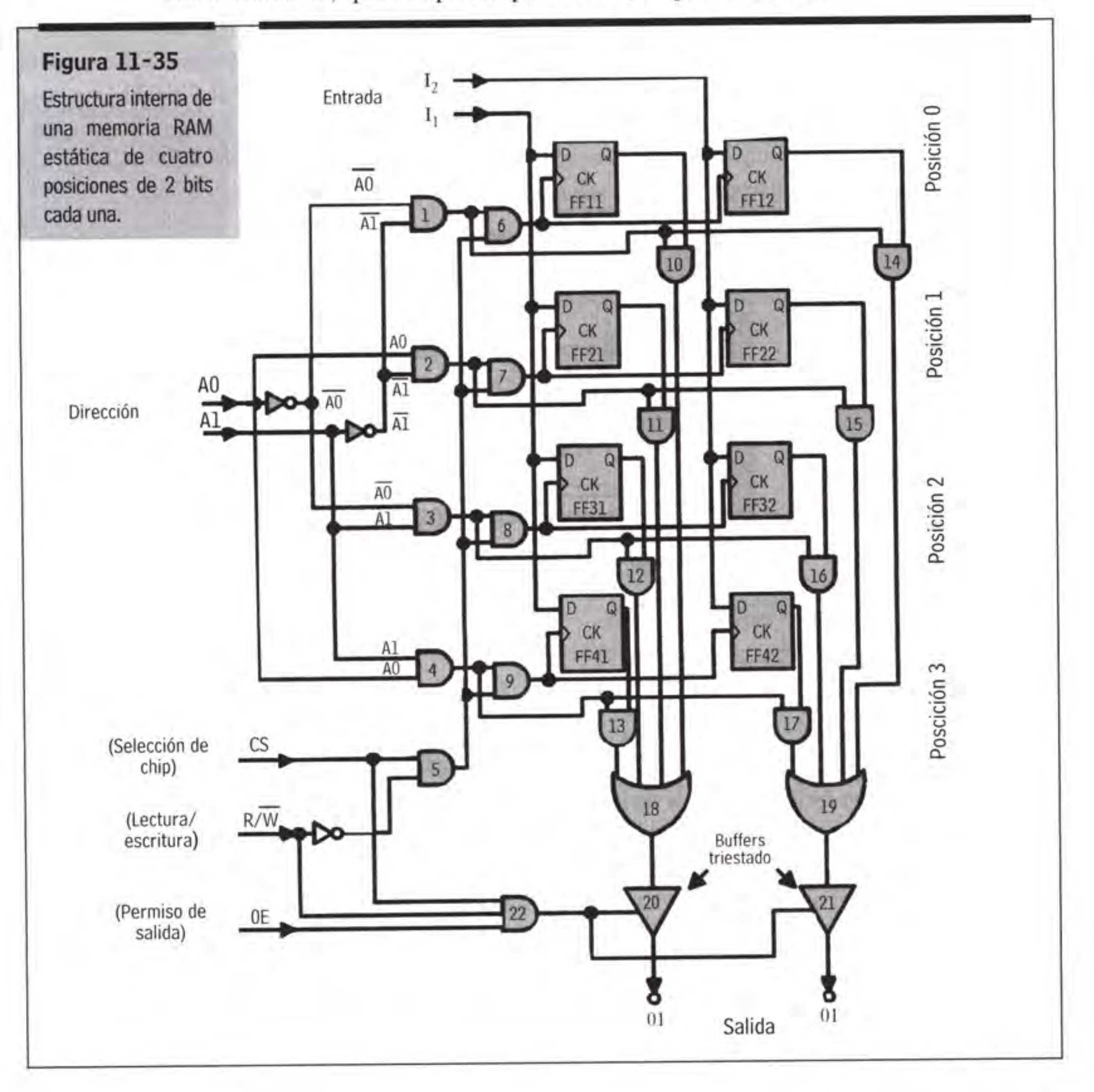


Figura 11-35
Estructura interna de una memoria RAM estática de cuatro posiciones de 2 bits cada una.

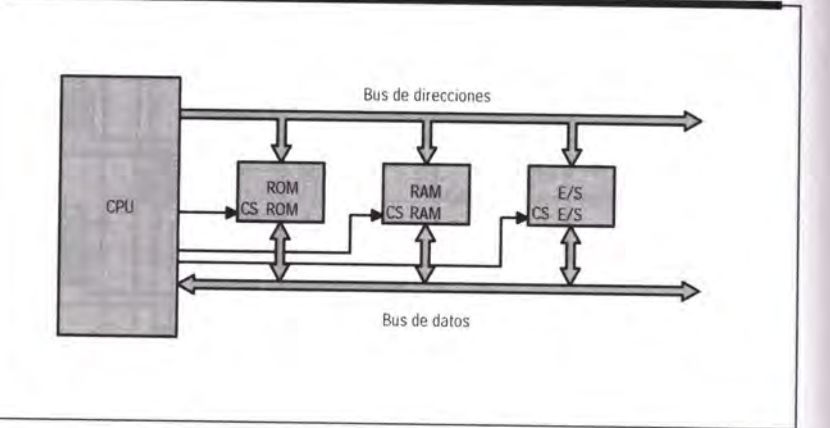
En un ciclo de lectura, la AND5 produce una salida de nivel bajo y no se genera impulso de reloj para los flip-flops. Cuando la posición seleccionada es la 2, una de las entradas de las puertas AND1 y AND15 recibe un 1, mientras que la otra recibe el contenido de FF31 y FF32, respectivamente. Si alguno de dichos flip-flops almacenaba un 1, obliga a la puerta AND correspondiente a sacar un 1 y aplicarlo como entrada a las puertas OR18 y OR19, que producen un nivel alto en las líneas de salida O1 y O2, respectivamente, cuando están activados los buffers 20 y 21. La activación de dichos buffers triestado se produce cuando AND22 saca nivel alto por su salida, cosa que sucede cuando CS = 1 y $R/\overline{W} = 1$ y hay permiso para la salida de información (OE = 1).

La señal CS de selección de chip, procedente de la CPU, es muy importante porque realiza dos funciones primordiales en la operatividad de la memoria:

1ª) Cuando existen varios módulos de memoria y dispositivos a enviar y recibir información al/desde el bus de datos, se selecciona el adecuado mediante CS. Los módulos y dispositivos no seleccionados quedan con sus salidas en alta impedancia. En la figura 11-36, si la CPU activa la señal CSRAM en una operación de lectura, debe ser el módulo de RAM el que *vuelque* la información contenida en la posición direccionada sobre el bus de datos, quedando en alta impedancia las salidas de los módulos de ROM y de E/S.

Figura 11-36

El único módulo que envía o recibe información al bus de datos es el que tiene activada su señal CS. En general, dicha señal procede de la decodificación de algunas líneas del bus de direcciones.



2ª) Desde que se activa CS hasta que son válidos los datos, transcurre un tiempo y desde que se desactiva CS hasta que dejan de ser válidos los datos, transcurre otro tiempo. En la figura 11-37 se muestra el diagrama de tiempos de un ciclo de lectura de una memoria RAM. En él se aprecia que, desde que se deposita una nueva dirección sobre el bus de direcciones hasta que la memoria proporciona el dato, transcurre un tiempo llamado **tiempo de acceso**. Una vez que se desactiva CS, el dato permanece válido un cierto tiempo.

En un ciclo de escritura, tras la activación de las señales CS = 1 y $R/\overline{W} = 0$, la CPU deposita en el bus de datos la información a grabar, en donde permanece válida durante un cierto tiempo. Figura 11-38.

Figura 11-37

Diagrama de tiempos de un ciclo de escritura de una memoria SRAM.

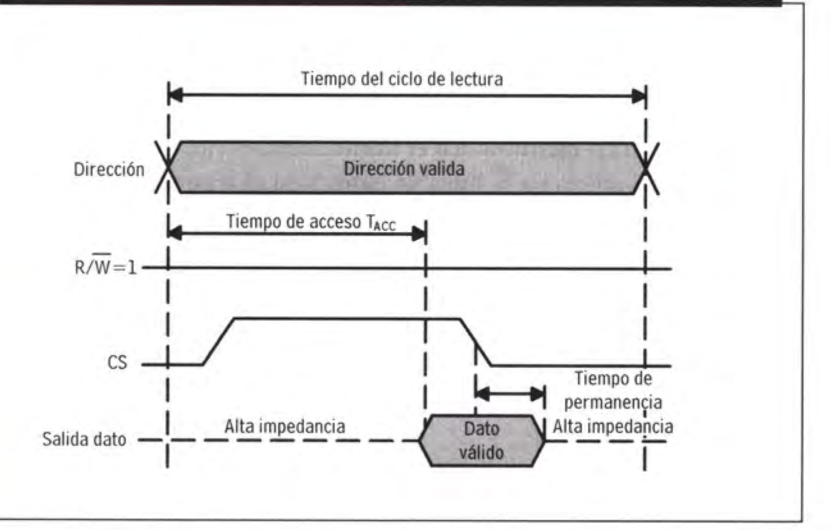
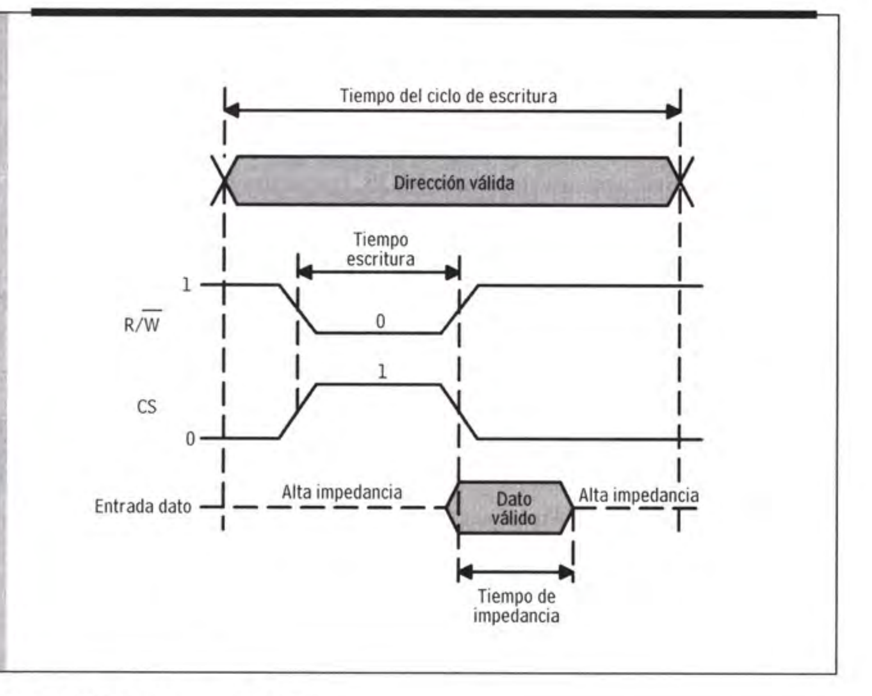


Figura 11-38

Cronograma de un ciclo de escritura de una SRAM.



11.5.2. RAM dinámicas o DRAM

En las memorias estáticas, la grabación de los bits en las celdas quedaba determinada por la conducción de uno de los transistores del flip-flop; en las memorias dinámicas el soporte físico del bit almacenado lo constituye un condensador, cuyo estado de carga representa el nivel lógico almacenado. Así, el 0 está implementado por la ausencia de carga en el condensador, mientras que el 1 lo está por su estado contrario.

La celda básica de una memoria dinámica se muestra en la figura 11-39, en la que se aprecia la presencia de un condensador gobernado por un interruptor que se controla mediante la línea de selección. Cuando se cierra el interruptor, comunica el condensador con la línea de datos al realizar una operación de lectura o escritura. En el primer caso, el valor del voltaje de carga del condensador se refleja en la línea de datos y en el segundo, el voltaje existente en esta última línea carga el condensador.

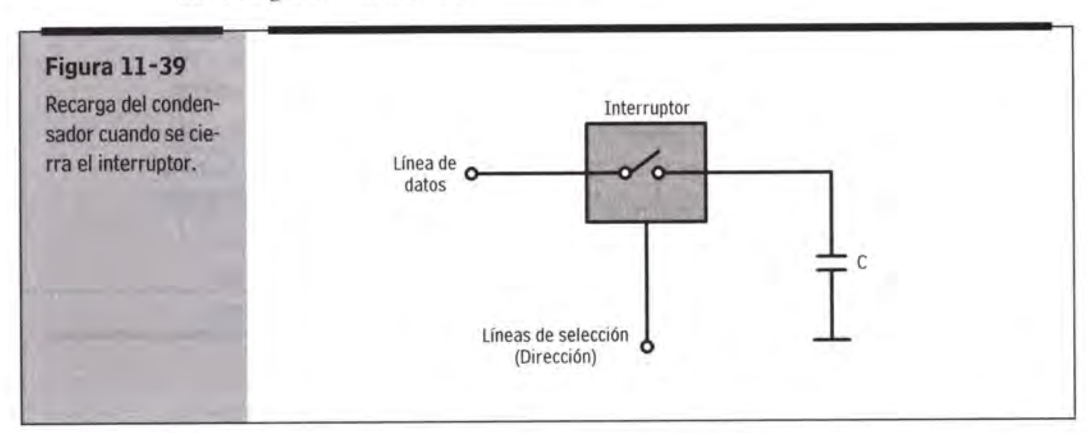


Figura 11-39
Recarga del condensador cuando se cierra el interruptor.

En la práctica, la implementación de una celda de memoria dinámica se realiza mediante un transistor MOS, que trabaja como interruptor y utiliza una capacidad parásita entre sus electrodos como elemento de almacenamiento de carga. La figura 11-40 presenta el esquema de este tipo de celda.

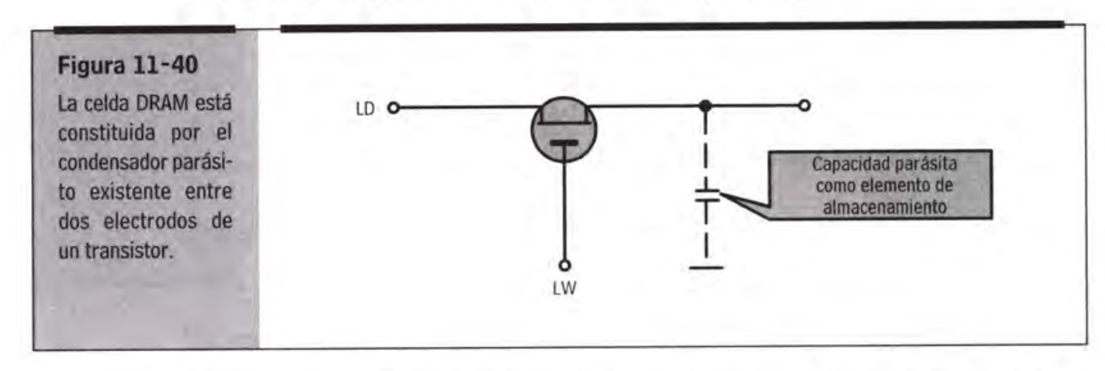


Figura 11-40
La celda DRAM está constituida por el condensador parásito existente entre dos electrodos de un transistor.

*El inconveniente de la celda formada por un condensador reside en la pérdida de la carga almacenada, debido a la corriente de fugas entre las armaduras del condensador. Para solventar este problema, se utiliza un **circuito de refresco**, cuya misión consiste en reponer cada muy poco tiempo la carga perdida por cada celda. El "ciclo de refresco", que continuamente precisan estas memorias, ha dado origen a la denominación de **dinámicas**.*

La enorme sencillez de la celda dinámica contrasta con el problema del refresco y el de la lectura destructiva. Al efectuar la lectura de la celda, el condensador pierde su carga, debido a su reducida capacidad con respecto a la línea de datos.

Por este motivo, hay que emplear una celda más compleja, basada en tres transistores, como se muestra en la figura 11-41.

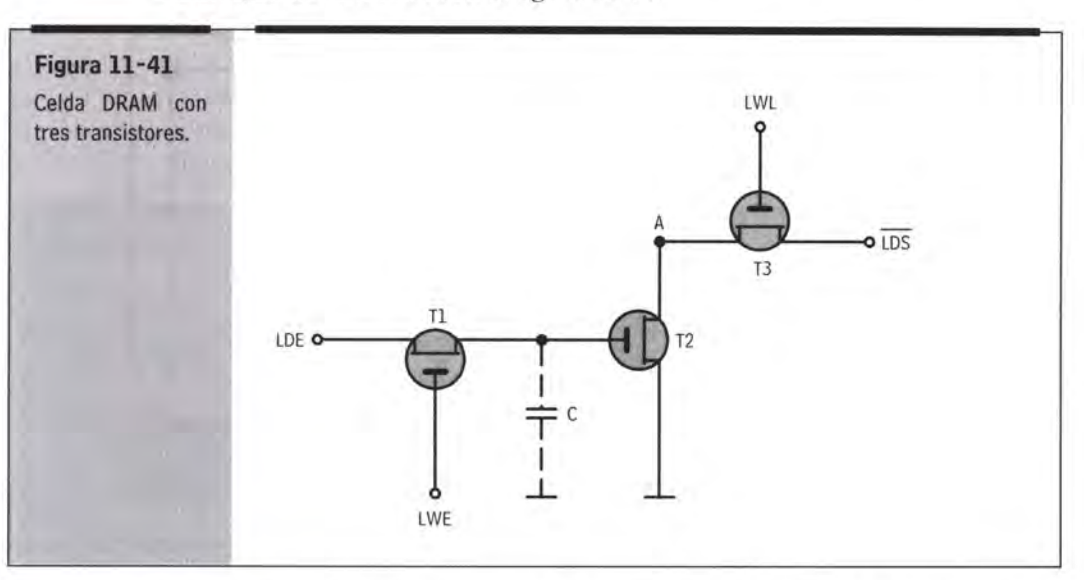


Figura 11-41
Celda DRAM con tres transistores.

La capacidad C, que aparece en la figura 11-41, corresponde a la que ofrece el Graduador del transistor T2 con respecto a masa. Se puede acceder a C por dos caminos independientes, según se desee leer o escribir la celda. La operación de lectura está controlada por T1 y la de escritura por T3.

Ciclo de escritura: a través de la línea de entrada de datos LDE, se envía la información a grabar al mismo tiempo que, por la línea de selección de entrada LWE, se manda un impulso de tensión que hace conducir a TI, cargando o descargando C según el nivel procedente de LDE.

Ciclo de lectura: se envía un impulso de tensión por la línea de selección de lectura, LWL, que provoca la conducción de T3. En la línea de datos de salida, $\overline{\text{LDS}}$, se obtiene un nivel de tensión opuesto al existente en C. En efecto, si en C hay un 1, o sea, el condensador está cargado, conducirá T2 y el punto A quedará a 0 V, que es la tensión transferida por $\overline{\text{LDS}}$. Cuando hay grabado un 0 en C, T2 está bloqueado y en A existe un nivel lógico alto que aparece en $\overline{\text{LDS}}$.

En general, la estructura interna de las DRAM agrupa las celdas en forma matricial, realizándose el acceso a las mismas mediante la selección de la fila y la columna adecuadas. En la figura 11-42 se presenta la arquitectura de una DRAM de 64K posiciones de 1 bit cada una con sus decodificadores de filas y columnas.

Para reducir el número de patitas destinado al direccionamiento de las celdas, se multiplexan en el tiempo la dirección de la fila y la columna, es decir, se usan las mismas patitas para introducir la dirección de la fila y la de la columna. Para indicar cuándo se envía la dirección de la fila por las patitas compartidas, se activa una señal auxiliar denominada **RAS** (*Row Address Strobe*: Habilitación de la dirección de la fila). Otro tanto sucede con la señal CAS (*Column Address Strobe*: Habilitación de la dirección de la columna). Figura 11-43.

Figura 11-42

En las DRAM, la selección de la celda de memoria se lleva a cabo mediante el direccionamiento de la fila y la columna en que se encuentra.

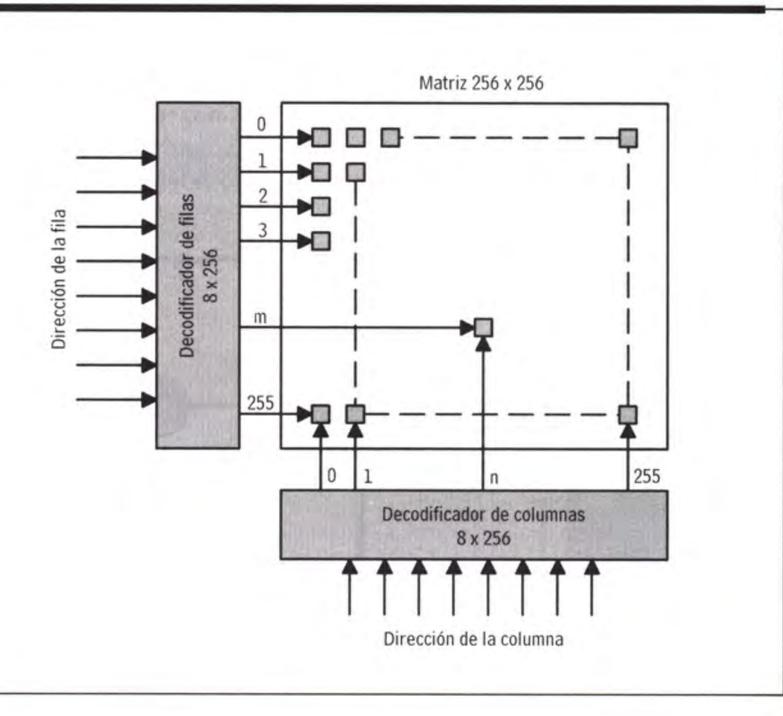
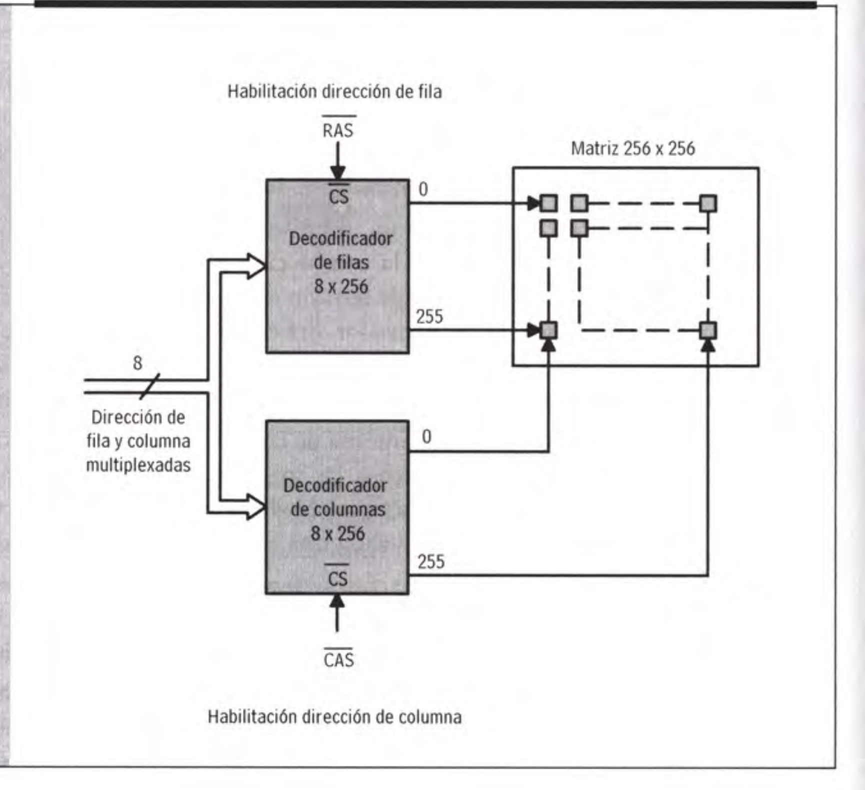


Figura 11-43

Para disminuir el número de patitas destinadas al direccionamiento de las DRAM, se multiplexan las líneas que llevan la dirección de la fila con las que llevan la dirección de la columna. Para reconocer cuándo está presente la dirección de cada tipo, se usan las señales CAS y RAS.



Utilizando el multiplexado de las direcciones de las filas y las columnas, una DRAM de 64K x 1 puede comercializarse en una pastilla con sólo 16 patitas. En la figura 11-44 se ofrece el diagrama de conexionado de la DRAM, modelo 4164, y en la 11-45 su estructura interna. Este modelo de memoria consume 275 mW y tiene un tiempo de acceso de 200 ns. Nótese que la matriz de memoria está formada por 512 columnas y 128 filas, ya que una de las líneas de dirección de fila se desvía al decodificador de columnas.

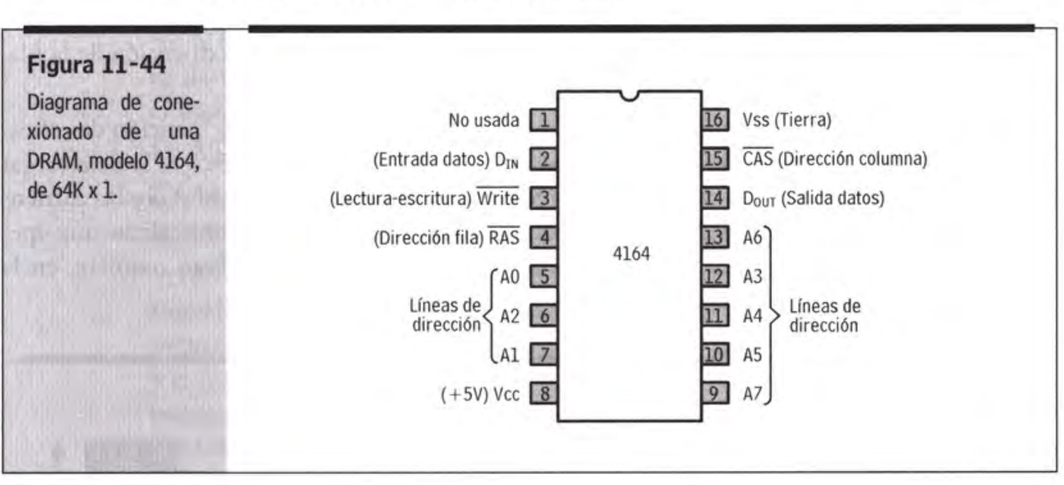


Figura 11-44
Diagrama de conexionado de una DRAM, modelo 4164, de 64K x 1.

Figura 11-45
Estructura interna simplificada de una DRAM 4164, con 65.536 celdas dispuestas en forma matricial, con 128 filas y 512 columnas.

Un bloque de memoria de 64K x 8 puede implementarse mediante ocho pastillas 4164 interconectadas como se refleja en la figura 11-46, estando gobernadas todas ellas por un **controlador de DRAM**. El contador de módulo 128 se encarga de realizar el refresco de las celdas de cada fila. Los ciclos pueden ser normales para la memoria o de refresco y su designación se hace a través de la señal M/$\overline{\text{R}}$. La señal $\overline{\text{RAS}}$ se activa al comienzo de cada ciclo de memoria y tras pasar por un circuito retardador, se convierte en la $\overline{\text{CAS}}$. Con las señales $\overline{\text{RAS}}$, $\overline{\text{CAS}}$ y M/$\overline{\text{R}}$, así como las de direcciones A0-A15 y el valor del contaje R0-R6, el multiplexor de la figura 11-46 saca en sus ocho líneas de salida la dirección de la fila, columna o fila de refresco apropiada.

Ciclo de refresco: se inicia cada 30 μs al activarse la señal de "petición de refresco" del temporizador usado para tal fin. La función del refresco es la de recargar cada celda de la memoria para recuperar las pérdidas ocasionadas por las corrientes de fuga. El refresco de una celda consiste, simplemente, en realizar una operación de lectura sobre ella antes de que pierda la carga. Esto requiere, en la mayoría de los casos, leer las celdas cada 2 ms como máximo.

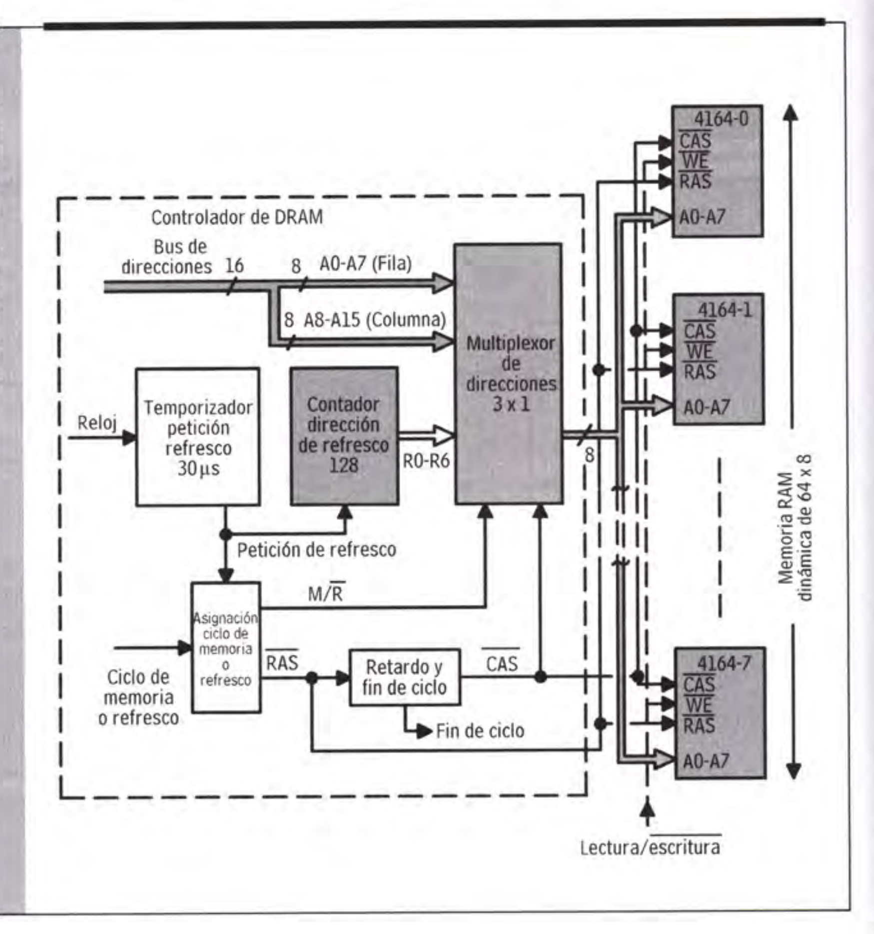


Figura 11-46

Esquema de un bloque de DRAM de 64K x 8, conformado por ocho pastillas 4164 y una lógica auxiliar encargada del refresco y el demultiplexado de las direcciones de las filas y columnas.


CADIZ
PUBLICA

La lectura de las celdas para refrescarlas se hace por filas, o sea, se leen todas las celdas de una fila simultáneamente. Los circuitos de control de la DRAM reciben la señal de refresco junto a la dirección de la fila y la activación de $\overline{CAS}$, procediendo automáticamente a leer o refrescar a todas las celdas de esa fila. En el modelo 4164 existen 128 filas, que se van refrescando secuencialmente cada 30 μs. La dirección de la fila a refrescar la proporciona el contador de módulo 128 por sus salidas R0-R6. Cada vez que se genera una petición de refresco, se incrementa el contador, que pasa a apuntar la siguiente fila.

En la figura 11-47 se muestra el cronograma de un ciclo de lectura en el circuito de memoria de la figura 11-46. El multiplexor selecciona una de sus tres entradas:

1ª) A0-A7 (Fila).

2ª) A8-A15 (Columna).

3ª) R0-R6 (Refresco de fila) de acuerdo con el estado de las señales de control $M/\overline{R}$ (Memoria/Refresco) y $\overline{CAS}$.

Inicialmente, el multiplexor selecciona la fila a acceder que se introduce por las señales A0-A7 cuando se activa $\overline{RAS}$ y $M/\overline{R} = 1$. Tras un retardo, se activa $\overline{CAS}$ y el multiplexor saca a los módulos DRAM la dirección de la columna, al proporcionar en su salida el valor A8-A15. Una vez que ha recibido cada chip de DRAM la dirección de la fila y la columna, deposita sobre el bus de datos la información de la celda seleccionada.

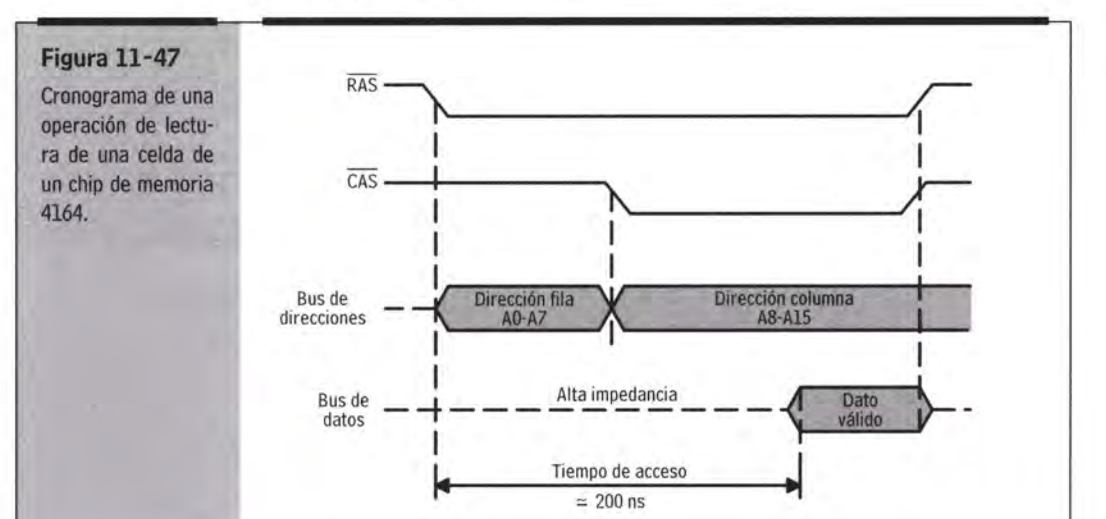


Figura 11-47
Cronograma de una operación de lectura de una celda de un chip de memoria 4164.

Existen modelos comerciales de memorias DRAM con una capacidad de varios Megabits. La tabla ofrecida en la figura 11-48 presenta los modelos de memoria RAM junto a sus características más importantes, que fabrica **Hitachi**.

Figura 11-48

Modelos de memorias MOS dinámicas y estáticas, fabricadas por HITACHI. Cortesía de AMITRON S.A.

Tipo	Capacidad	Configuración	Proceso	Código	Sufijo	Características (máx.)		Empaquetamiento									Observaciones
						Tiempo acceso (ns)	I_{CC} (mA) Active/Standby	Nº Pimes	G-DIP	P-DIP	SK-DIP	LCC	PLCC	SOP	FPP	Otras	
RAM DINÁMICAS	64K	64K x 1	N	HM 4864 P	- 3	200	60/3.5	16	○	●							
					- 2	150		16	○	●							
				HM 4864 AP	- 20	200	40/5.5	16		○		○					
					- 15	150	50/5.5	16		○		○					
					- 12	120	55/5.5	16		○		○					
		16K x 4	N	HM 48416 AP	- 20	200	45/3.5	18		○							
					- 15	150	55/3.5	18		○							
					- 12	120	60/3.5	18		○							
	256K	156K x 1	N	HM 50256 P	- 20	200	55/4.5	16	○	○			○			○	
					- 15	150	70/4.5	16	○	○			○			○	
					- 12	120	83/4.5	16	○	○			○			○	
				HM 50257 P	- 20	200	55/4.5	16	○	○			○				Nibble mode
					- 15	150	70/4.5	16	○	○			○				
					- 12	120	83/4.5	16	○	○			○				
			C	HM 51256 P	- 15	150	40/2	16		○							Page mode
					- 12	120	50/2	16		○							
					- 10	100	60/2	16		○							
					- 8	85	70/2	16		○							
				HM 51256 LP	- 15	150	40/0.3	16		○							Page mode
					- 12	120	50/0.3	16		○							
					- 10	100	60/0.3	16		○							
					- 8	85	70/0.3	16		○							
				HM 51258 P	- 15	150	40/2	16		○							Columna mode
					- 12	120	50/2	16		○			○				
					- 10	100	60/2	16		○			○				
					- 8	85	70/2	16		○			○				
		64K x 4	N	HM 50464 P	- 20	200	55/4.5	18		○			○				Page mode
					- 15	150	70/4.5	18		○			○				
					- 12	120	83/4.5	18		○			○				
				HM 50465 P	- 20	200	55/4.5	18		○							Nibble mode
					- 15	150	70/4.5	18		○							
					- 12	120	83/4.5	18		○							
			C	HM 53461 P	-15	150	85/5	24		○							
					- 12	120	105/5	24		○							Multi-port RAM
					- 10	100	125/5	24		○							
				HM 53462 P	- 15	150	85/5	24		○							Multi-port RAM with logic operating function
					- 12	120	105/5	24		○							
					- 10	100	125/5	24		○							
	1M	1M x 1	C	HM 511000	- 15	150	50/2	18	○								Page mode
					- 12	120	60/2	18	○								
					- 10	100	70/2	18	○								
				HM 511001	- 15	150	50/2	18	○								Nibble mode
					- 12	120	60/2	18	○								
					- 10	100	70/2	18	○								
	Module	256K x 4		HB 561004 A	- 20	200	220/18	22								○	SIP Module (lead type I/O separated) with 4 pcs of HM 50256 CP mounted
					- 15	150	280/18	22								○	
					- 12	120	330/18	22								○	
		256K x 5	N	HB 561005 A	- 20	200	275/23	24								○	SIP module (lead type, I/O separated) with 5 pcs of HM 50256 CP mounted
					- 15	150	350/23	24								○	
					- 12	120	413/23	24								○	
		256K x 8	N	HB 561008 B	- 20	200	440/36	30								○	SIP module (socket type, I/O common) with 8 pcs of HM 50256 CP mounted
					- 15	150	560/36	30								○	
					- 12	120	660/36	30								○	
		256K x 9	N	HB 561003 A	- 20	200	495/40	30								○	SIP module (lead type, I/O common) with 9 pcs of HM 50256 CP mounted
					- 15	150	630/40	30								○	
					- 12	120	747/40	30								○	
				HB 561003 B	- 20	200	495/40	30								○	SIP module (socket type, I/O common) with 9 pcs of HM 50256 CP mounted
					- 15	150	630/40	30								○	
					- 12	120	747/40	30								○	
RAM ESTÁTICAS	4K	4K x 1	C	HM 6147 HP	- 55	55	80/0.8	18	○	○							
					- 45	45	–	18	○	○							
					- 35	35	–	18	○	○							
				HM 6147 HLP	- 55	55	80/0.1	18		○							
					- 45	45	–	18		○							
					- 35	35	–	18		○							
		1K x 4	C	HM 6148 HP	- 55	55	100/0.8	18	○	○							
					- 45	45	–	18	○	○							
				HM 6148 HLP	- 55	55	100/0.5	18		○							
					- 45	45	–	18		○							
	16K	16K x 1	C	HM 6167 P	- 8	100	60/2	20	○	○							
					- 6	85	–	20	○	○							
					–	70	–	20	○	○							
				HM 6167 LP	- 8	100	60/0.05	20		○							
					- 6	85	–	20		○							
					–	70	–	20		○							
				HM 6167 HP	- 55	55	80/2	20	○	○		○					
					- 45	45	–	20	○	○		○					
				HM 6167 HLP	- 55	55	80/0.05	20		○							
					- 45	45	–	20		○							
				HM 6267 P	- 45	45	80/2	20		○		○					
					- 35	35	100/2	20		○		○					
		4K x 4	C	HM 6168 HP	- 70	70	90/2	20		○							
					- 55	55	–	20		○							
					- 45	45	–	20		○							
				HM 6168 HLP	- 70	70	90/0.05	20		○							
					- 55	55	–	20		○							
					- 45	45	–	20		○							
			C	HM 6268 P	- 35	35	TBD	20		○							
					- 25	25	–	20		○							

 © ITES - PARANINFO

11.5.3. Aplicaciones

La principal aplicación de las memorias RAM radica en su uso como memoria principal en los computadores; se emplean muchos módulos en una placa, junto a los circuitos de control y tiempos. Las tarjetas de memoria se ensamblan para formar bancos de memoria de muchos megabits.

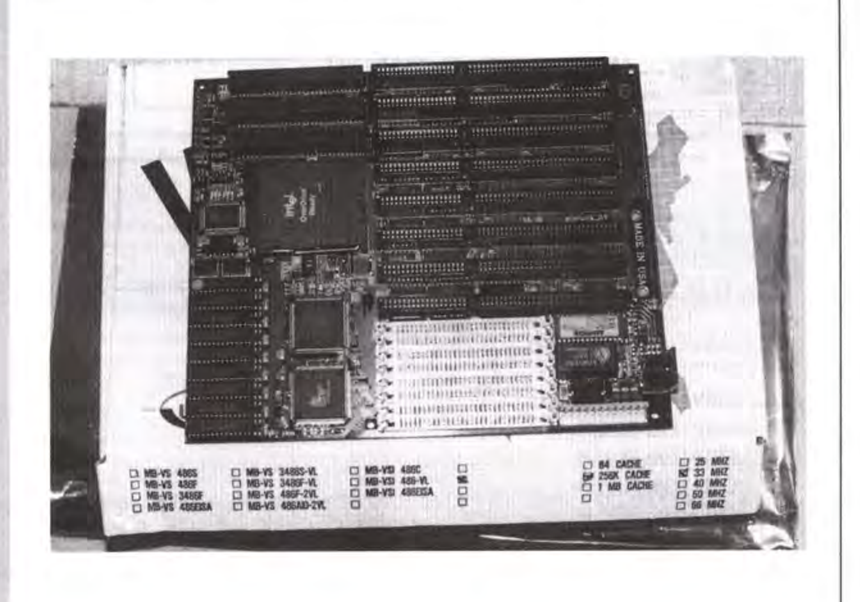
Para el almacenamiento de microprogramas en los computadores, se emplean los chips de RAM para contener ciertas partes del programa (microinstrucciones). En este caso, el programa principal no dispone de aquellas microinstrucciones y permite a la Unidad de Control sacarlas internamente a mayor velocidad.

También se usan las RAM en los terminales remotos de un computador, en los que se realizan diversas operaciones con los datos, de forma similar al propio computador. Efectúan cierto número de operaciones secundarias en el terminal y después pasan al computador para ser sometidos a un posterior procesamiento. El empleo del término **terminal inteligente** se debe a la posibilidad de la RAM de almacenar programas y ejecutar instrucciones mediante un microprocesador.

En la figura 11-49 se muestra la fotografía de la placa principal de un microcomputador AT, en la que la mayor parte de los chips están dedicados a conformar la memoria principal compuesta por un banco de RAM y otro de ROM.

Figura 11-49

Fotografía de la placa principal de un microcomputador AT. La mayoría de sus chips implementan la memoria principal.

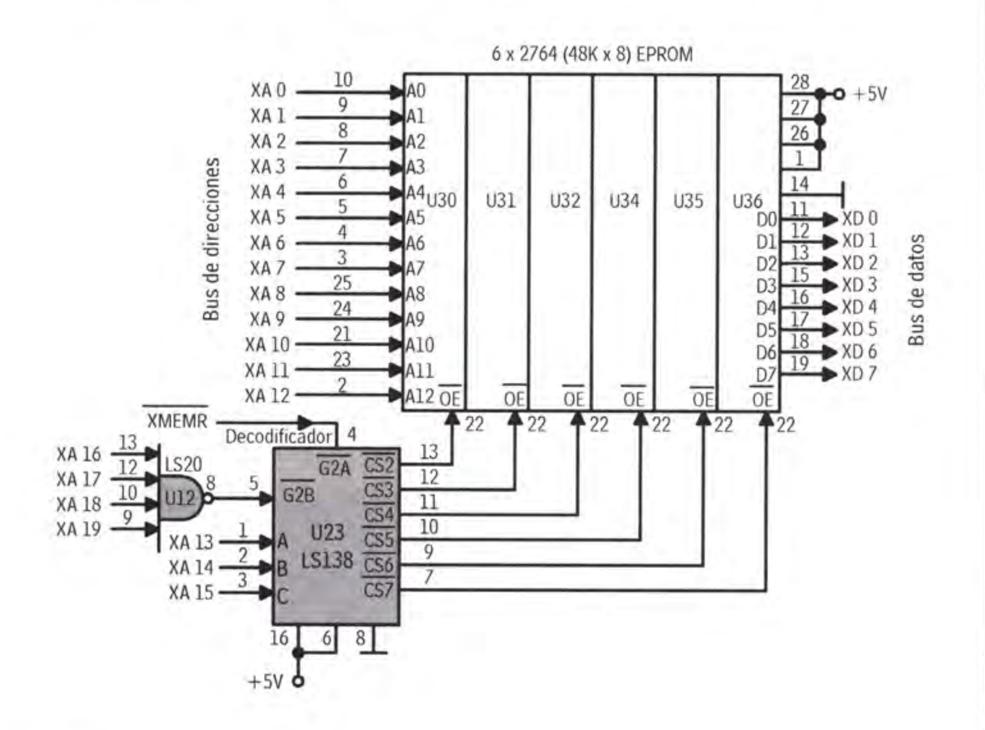


Ejemplo 11-9

A) Un microprocesador 8086 dispone de un bus de direcciones de 20 líneas (A0-A19) y tiene conectado un banco de memoria formado por seis chips de EPROM, modelo 2764, con un tamaño de 8K x 8 cada uno. En la figura 11-50 se muestra un esquema simplificado del mencionado banco, que es direccionado por el decodificador LS138(U23) de tres entradas (A, B y C) y ocho salidas ($\overline{CS0}$-$\overline{CS7}$). Deducir la dirección inicial a la que responde cada pastilla 2764.

Figura 11-50

Banco de memoria EPROM compuesto por seis pastillas 2764 de 8K x 8, direccionada por el decodificador 74LS138 de tres entradas y ocho salidas.



SOLUCIÓN

CHIP U30: F400 H.

CHIP U31: F600 H.

CHIP U32: F800 H.

CHIP U34: FA00 H.

CHIP U35: FC00 H.

CHIP U36: FE00 H.

Ejemplo 11-10

B) Con referencia al bloque de memoria representado en la figura 11-51, contestar las siguientes preguntas:

a) ¿Cuál es la configuración total de la memoria?

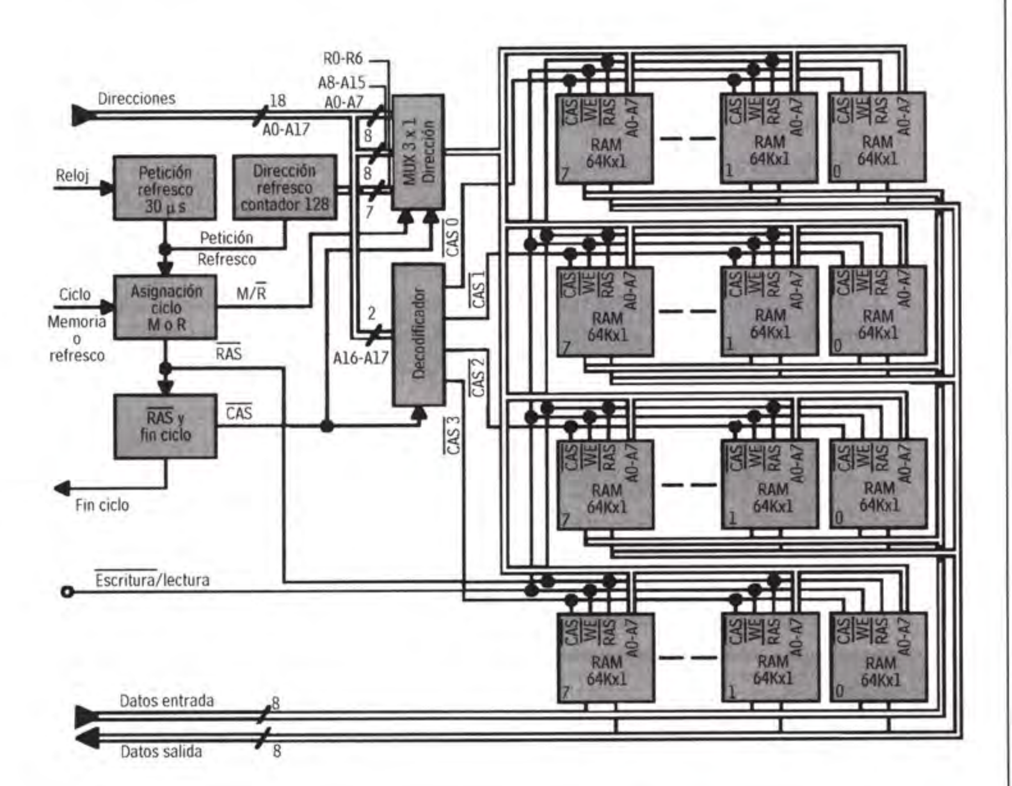
b) ¿Cuál es la función de la señal $\overline{CAS0}$?

c) ¿Cada cuánto tiempo se refrescan las celdas de cada fila de los chips de DRAM?

d) ¿A qué direcciones responde el banco inferior de la memoria controlado por la señal $\overline{CAS3}$?

Figura 11-51

Esquema de un bloque de memoria DRAM.



SOLUCIÓN

a) 256K x 8.

b) Proporcionar la dirección de la columna a los ocho chips de DRAM del banco superior de 64K x 8.

c) 128 x 30 = 3.840 μs.

d) A las 64K direcciones de la memoria que tienen A16 = A17 = 1.

11.6. OTROS TIPOS DE MEMORIAS

En este apartado se describen someramente algunos tipos de memorias cuyas características y aplicaciones son similares a las de los semiconductores. No se hace referencia a las memorias secundarias o masivas, como discos, cintas magnéticas y discos ópticos, que, aunque poseen una elevada capacidad y un bajo coste por bit, al ser del tipo de acceso secuencial tienen un tiempo de acceso medio muy elevado y no se pueden usar como memoria principal del computador.

11.6.1. Memorias asociativas (caché)

En este tipo de memoria se realiza la búsqueda de información por su propio contenido y no por la dirección en la que se ubica. Se les llama **CAM** (Memorias de Acceso por Contenido). En lugar de proporcionar la dirección de la posición a acceder, se dispone de una información, denominada *descriptor*, que se compara con los contenidos de las posiciones para determinar la posición a acceder.

En la operación de escritura de las memorias CAM se graba en la posición el descriptor junto al dato que lleva asociado. También se puede efectuar la escritura realizando una lectura previa para localizar la existencia de alguna posición con el mismo descriptor que el que se intenta escribir. Si existe dicho descriptor, sólo se reemplazará la información que lleve asociada y, en caso negativo, se grabarán el descriptor y la información asociada en una posición libre de la memoria.

Las CAM son memorias muy rápidas y eficaces para el manejo de tablas. También se suelen usar como memorias caché en los computadores, en cuyo caso actúan como memorias intermedias entre la memoria principal y la CPU, disminuyendo considerablemente el tiempo de acceso.

En la figura 11-52 se presenta la estructura general de una memoria CAM a la que se accede según el contenido de los descriptores. Una vez que se ha localizado la posición que dispone del descriptor buscado, se lee o se escribe la información que lleve asociada.

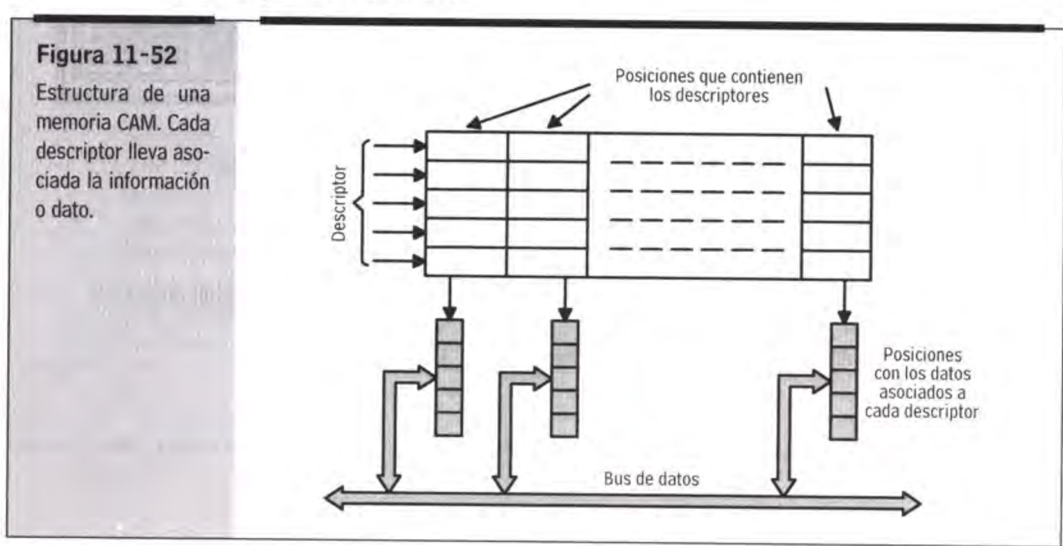


Figura 11-52
Estructura de una memoria CAM. Cada descriptor lleva asociada la información o dato.

11.6.2. Memorias de acceso secuencial

Son memorias de lectura y escritura que difieren sustancialmente de las RAM en la forma de acceder a la información de las celdas de memoria.

En las memorias de acceso secuencial para leer o escribir una celda, previamente hay que pasar por todas las celdas que la preceden. Se trata de un tipo de direccionamiento semejante al empleado en las cintas magnéticas.

Las memorias de acceso secuencial son eficaces en aquellos casos en los que hay que acceder a gran número de datos secuenciales, de forma periódica, como sucede en la visualización de caracteres en las pantallas.

Dentro de las memorias con semiconductores hay dos tipos de memorias secuenciales: *los registros de desplazamiento,* fabricados con tecnología bipolar y MOS, y *las memorias de acoplo de carga CCD*, de tecnología MOS.

En general, las memorias secuenciales están organizadas en varios bucles, como se refleja en la figura 11-53.

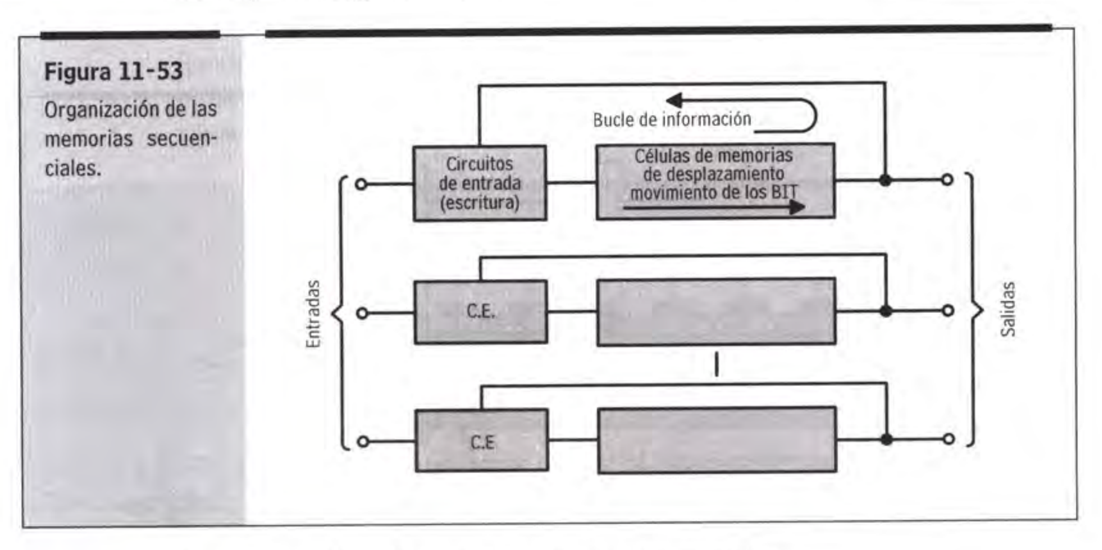


Figura 11-53
Organización de las memorias secuenciales.

La estructura basada en bucles presenta dos ventajas:

1ª. En caso de precisar una información de palabras de n bits, basta utilizar **n** bucles en paralelo.

2ª. Cuando la información llega en serie, es posible acceder a una posición de la memoria más rápidamente, utilizando la estructura de bucles múltiples.

La ausencia de elementos móviles en estas memorias les confiere una neta superioridad en ciertos aspectos respecto a los discos y cintas magnéticos.

MEMORIAS SECUENCIALES BASADAS EN REGISTROS DE DESPLAZAMIENTO.

En la figura 11-54 se presenta el esquema que adoptan este tipo de memorias para configurar los bucles.

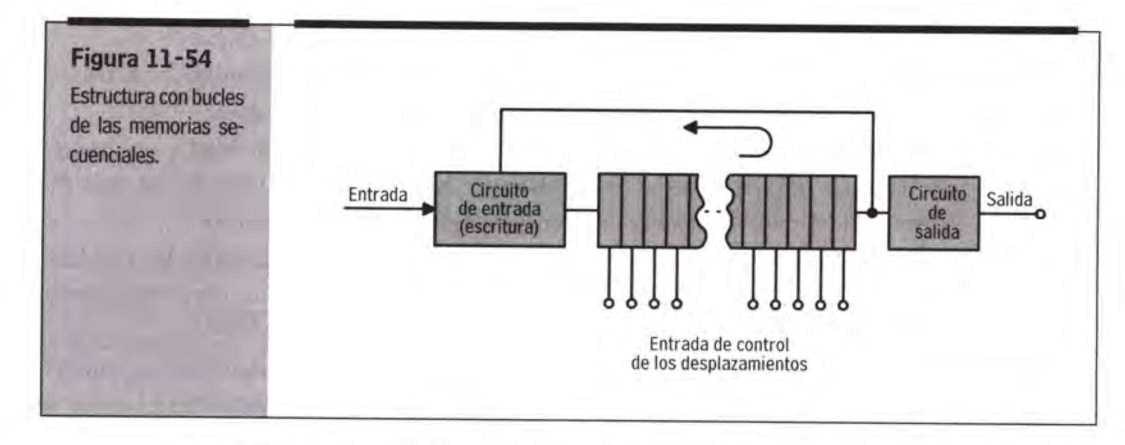


Figura 11-54
Estructura con bucles de las memorias secuenciales.

El comportamiento de estos dispositivos es síncrono y está controlado por dos impulsos de reloj, que tienen la forma mostrada en la figura 11-55. La información se traslada hacia la derecha, de una etapa a la siguiente, con la generación de dichos impulsos.

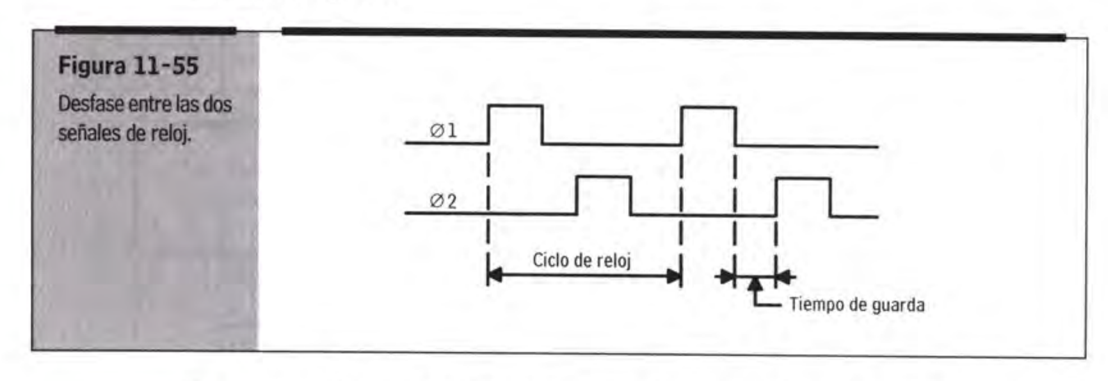


Figura 11-55
Desfase entre las dos señales de reloj.

La máxima frecuencia de trabajo viene determinada por la mínima duración que deben tener los impulsos para poder cargar completamente las capacidades en las que se almacena la información. La frecuencia mínima queda establecida por el tiempo máximo que puede mantenerse la información en los condensadores de almacenamiento.

Un registro de desplazamiento dinámico con transistores MOS puede representarse, de forma simplificada, como aparece en la figura 11-56, en donde la información se transmite de la siguiente manera:

a) Con el primer impulso Ø1, la información presente en la entrada pasa a C1. Si dicha información era 1, después que Ø1 = 1, C1 = 1.

b) Con el primer impulso Ø2, pasa a C2 el complemento de la información que había en Cl. En el caso de ejemplo, C2 = 0.

c) Con el segundo impulso de Ø1 = 1, el complemento de la información contenida en C2 pasa a C'1.

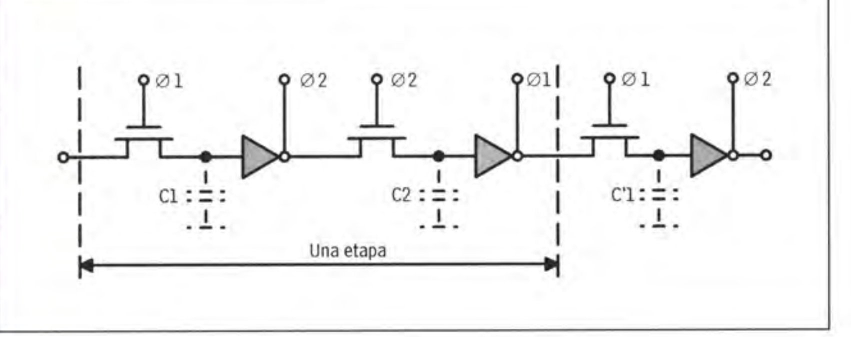


Figura 11-56
Registro de desplazamiento con transistores MOS.

Por tanto, transcurrido un periodo, la información se ha desplazado una etapa a la derecha.

MEMORIAS CCD, CON DISPOSITIVOS DE ACOPLO DE CARGA.

También en este tipo de memoria se utilizan los bucles sobre los que circula la información de una manera cíclica. La estructura de un dispositivo de acoplo de carga viene representada en la figura 11-57.

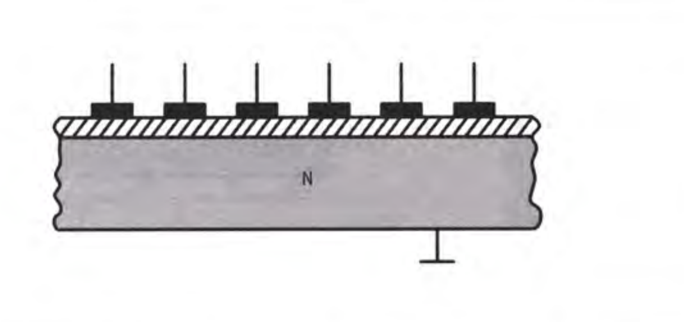


Figura 11-57
Estructura general de un dispositivo de acoplamiento de carga.

El sustrato mostrado en la figura 11-57 está formado a base de silicio semiconductor N o P. Está recubierto con una capa de SiO_2 aislante, sobre la que se hallan colocados una serie de electrodos metálicos con una separación muy pequeña entre ellos.

Si se aplica a los electrodos una tensión negativa -V (se supone que el substrato es de tipo N), cuya magnitud supere la tensión de umbral, se producirá una zona de despoblamiento de portadores mayoritarios (electrones), que serán expulsados, mientras son atraídos a dicha zona los minoritarios. La situación queda reflejada en la figura 11-58.

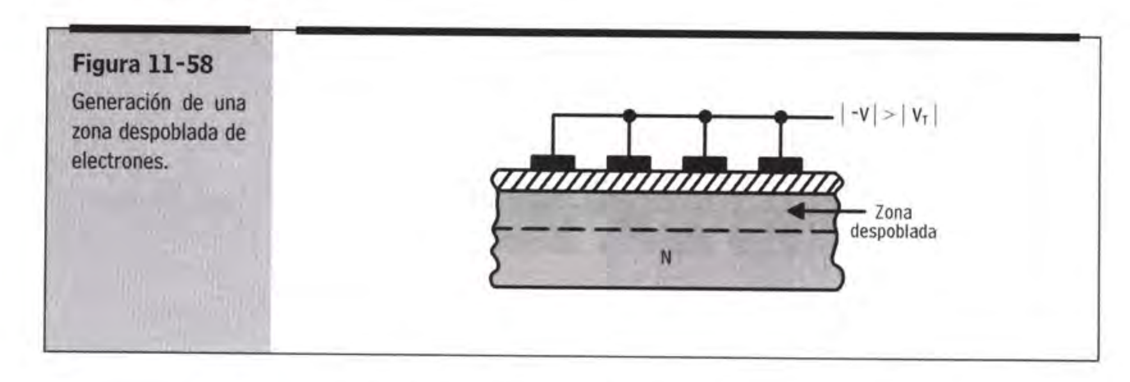


Figura 11-58
Generación de una zona despoblada de electrones.

En la figura 11-58 no se ha representado la finísima capa de carga positiva que se produce en la superficie inferior del SiO_2 y que es la causante de la neutralización del campo eléctrico causado por -V. Dichos huecos están atrapados por el campo eléctrico y, por tanto, inmóviles. Dependiendo de la magnitud de -V, la zona despoblada será mayor o menor. Si los electrodos no están a la misma tensión, se producen zonas despobladas como se muestra en la figura 11-59.

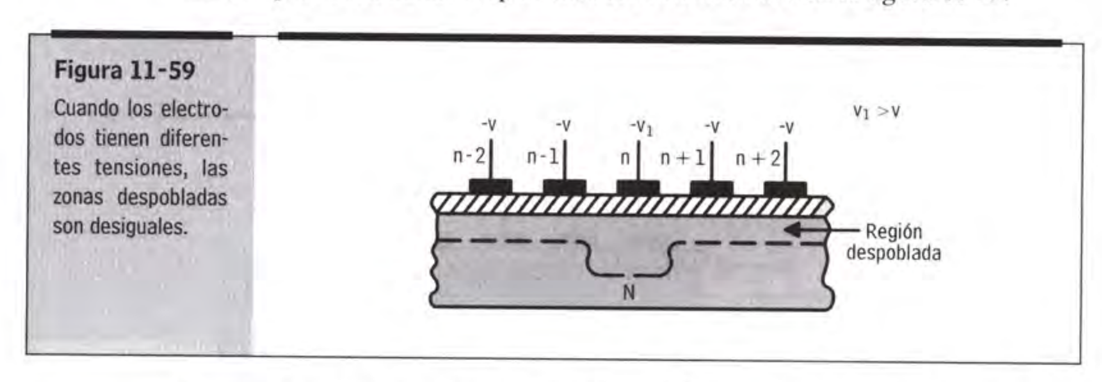


Figura 11-59
Cuando los electrodos tienen diferentes tensiones, las zonas despobladas son desiguales.

Se aprecia en la figura 11-59 que, debajo del electrodo al que se ha aplicado la mayor tensión, se origina una zona de despoblamiento superior. La amplitud de dicha zona depende de la tensión.

Si, mediante un método adecuado, se introduce una cierta carga positiva en la zona correspondiente al electrodo n conectado a -V1, permanecerá en dicha zona debido a que existe una barrera de potencial en sus dos lados (zona más positiva). Si se consigue trasladar dicha carga, antes de que desaparezca por difusión, hacia uno de sus lados, se obtendrá un resultado similar al conseguido con los registros de desplazamiento.

El procedimiento para desplazar las cargas en una determinada dirección consiste en la aplicación de unos impulsos de tensión en los distintos electrodos, de forma secuencial.

Una posible configuración referente a la interconexión de electrodos y las formas de onda de tensión queda representada en la figura 11-60.

Si se supone, por un momento, que debajo del electrodo **n**, de la figura 11-60, se ha introducido una carga positiva, y que dicho electrodo se encuentra inicialmente (t = 0) a una tensión $-V_2$, dicha carga no podrá moverse hacia la izquierda debido a que la tensión en el electrodo de la izquierda se está haciendo, en ese momento, más positiva, y, por lo tanto, son repelidas las cargas positivas. En el instante t_1, la tensión en el electrodo de la derecha (n + 1) alcanza el valor $-V_2$, mientras que en el electrodo **n** comienza a aumentar la tensión, lo que provoca el desplazamiento de las cargas positivas hacia la zona más negativa. Es decir, en el intervalo de tiempo $t_1 - t_2$, la carga se desplaza desde el electrodo **n** al **n + 1**. Análogamente ocurrirá a partir del instante t_2 en el que el electrodo n + 2, unido a ∅3, alcanza el valor $-V_2$, comenzando el electrodo **n + 1** a pasar de $-V_2$ a $-V_1$. Ambos efectos combinados ocasionan una nueva transferencia de carga hacia la derecha.

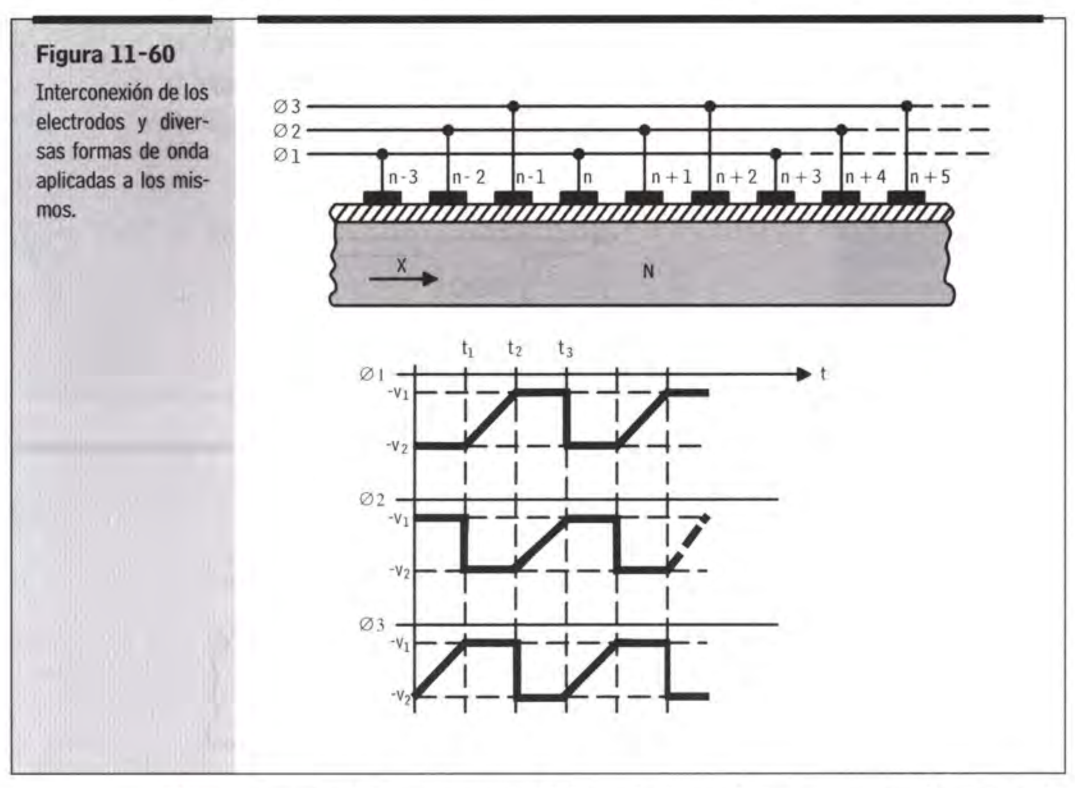


Figura 11-60
Interconexión de los electrodos y diversas formas de onda aplicadas a los mismos.

Una representación esquemática del movimiento de las cargas por los pozos de potencial se muestra en la figura 11-61.

Si las fases de reloj ∅1, ∅2 y ∅3 se detuviesen por cualquier causa, se perdería la información almacenada en unos pocos milisegundos. Éste es un grave inconveniente de este tipo de memorias.

Existen diversos sistemas para introducir la información (secuencia de bits 1 y 0) a los dispositivos CCD. Destacan los dos siguientes:

1º) Mediante la creación de pares electrón-hueco por efecto fotoeléctrico.

2º) Mediante una unión N-P delante del primer electrodo, según se muestra en la figura 11-62.

Figura 11-61

Representación gráfica del movimiento de las cargas dentro de los pozos de potencial.

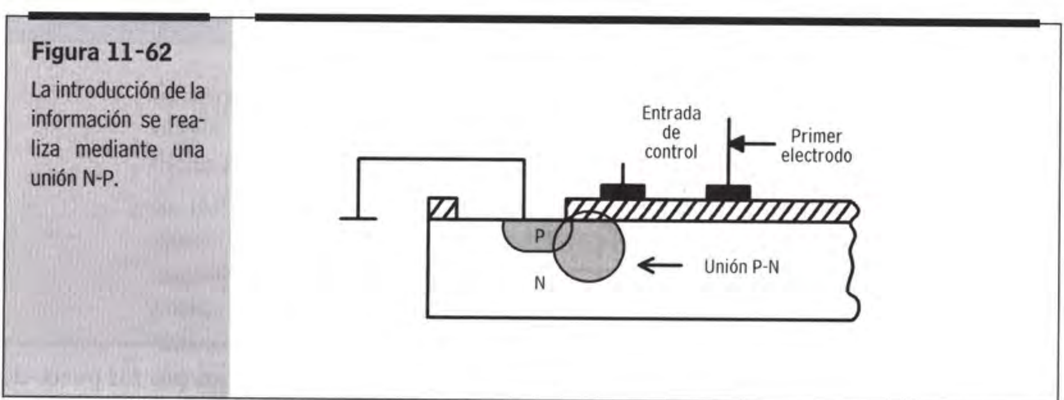


Figura 11-62

La introducción de la información se realiza mediante una unión N-P.

Para extraer la información se puede emplear una capacidad MOS o bien un diodo, tal como se representa en la figura 11-63.

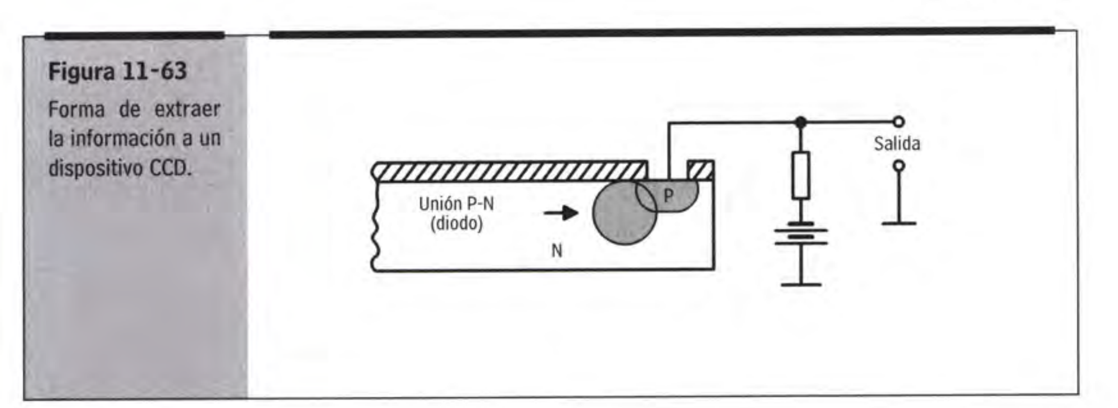
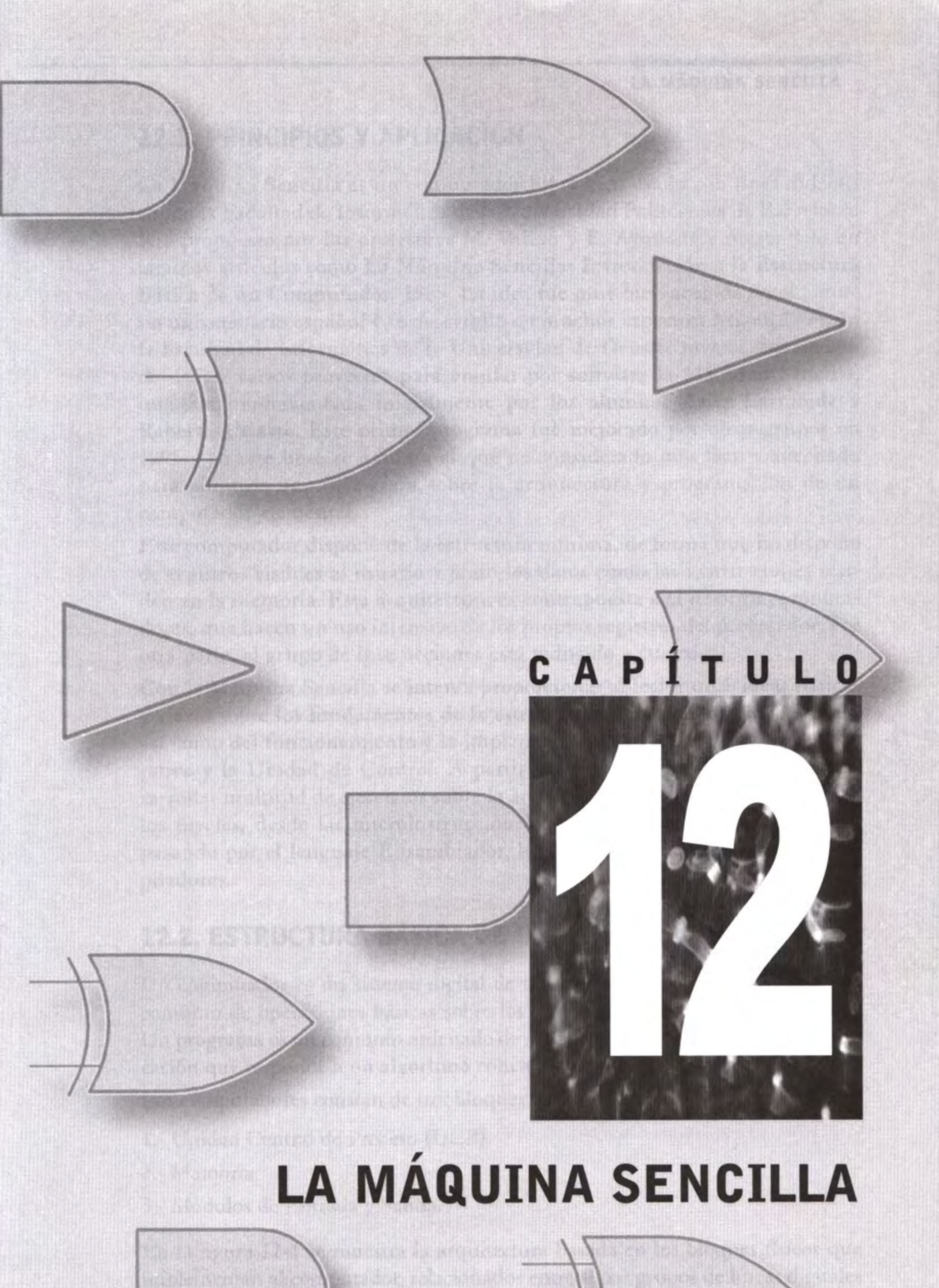


Figura 11-63
Forma de extraer la información a un dispositivo CCD.

Las memorias CCD tienen una velocidad semejante a las basadas en registros de desplazamiento, pero poseen dos características muy apreciadas en algunas aplicaciones:

a) Menor consumo de energía.

b) Mayor densidad de almacenamiento.



CAPÍTULO 12

LA MÁQUINA SENCILLA

12.1. PRINCIPIOS Y APLICACIÓN

La **Máquina Sencilla** es un computador hipotético usado con fines didácticos en la Facultad de Informática de la Universidad Politécnica de Barcelona. Fue propuesto por los profesores **M. Valero** y **E. Ayquadé** y presentado en algunos artículos como **La Máquina Sencilla: Introducción a la Estructura Básica de un Computador**, 1989. La idea fue muy bien acogida en el entorno universitario español y se desarrolló en muchos aspectos. Así, en 1994, en la Facultad de Informática de la Universidad de Deusto, tuve la satisfacción de dirigir varios proyectos para emular por software la Máquina Sencilla, que fue implementada inicialmente por los alumnos **Asier Larrabide** y **Roberto Uriarte**. Este primer programa fue mejorado por otros grupos en 1995 y en este libro se adjunta el que he considerado más fácil y adecuado para adquirir una idea clara sobre la arquitectura y programación de un computador elemental.

Este computador dispone de la estructura mínima, de forma que no dispone de registros visibles al usuario y tanto los datos como las instrucciones residen en la memoria. Esta arquitectura es contrapuesta a la de otros computadores, que hacen un uso intensivo de los propios registros del procesador. Por otra parte, el grupo de instrucciones está reducido a cuatro.

Con la Máquina Sencilla se intenta proporcionar al lector unas ideas básicas y claves sobre los fundamentos de la estructura interna de los procesadores, así como del funcionamiento y la implementación física de la Unidad Operativa y la Unidad de Control. A partir de estos conceptos se pueden desarrollar multitud de ejercicios sobre la arquitectura del sistema físico a todos los niveles, desde las microinstrucciones a las instrucciones de alto nivel, pasando por el lenguaje Ensamblador, los Sistemas Operativos y los Compiladores.

12.2. ESTRUCTURA BÁSICA DE UN COMPUTADOR

Un computador es un sistema digital de tipo secuencias capaz de efectuar un conjunto de operaciones básicas sobre los datos, que se llaman **instrucciones**. Un **programa** es un conjunto ordenado de instrucciones que resuelven una aplicación que responde a un **algoritmo** concreto.

Los computadores constan de tres bloques principales:

1.- Unidad Central de Proceso (UCP).

2.- Memoria.

3.- Módulos de Entrada y Salida.

En la figura 12-1 se muestra la arquitectura basada en los bloques físicos que implementan al computador, relacionados entre sí por grupos de líneas digitales que intercambian información binaria y que reciben el nombre de **buses**.

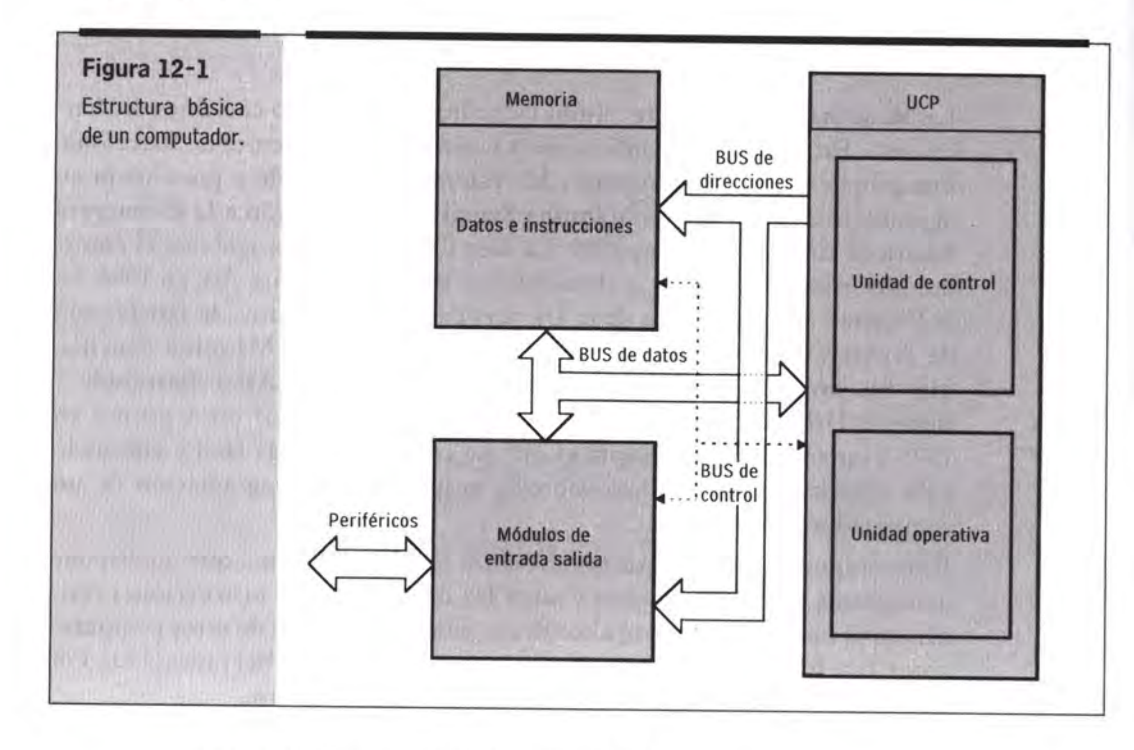


Figura 12-1
Estructura básica de un computador.

12.2.1. Unidad Central de Proceso (UCP)

Tiene la misión de interpretar las instrucciones que provienen del programa almacenado en la memoria y ordenar su ejecución mediante una secuencia de operaciones elementales llamadas **microinstrucciones**. También dispone de los circuitos necesarios para la realización de las principales operaciones, que suelen ser de tipo lógico y aritmético. Consta de dos partes:

a) Unidad de Control (UC)

Es la encargada de la interpretación de las instrucciones y la generación de las señales que controlan los restantes bloques del computador y llevan a cabo las microinstrucciones que componen cada instrucción.

b) Unidad de Proceso (UP)

Está diseñada para realizar las operaciones que implican las instrucciones sobre los datos. Por este motivo suele recibir el nombre de Camino de Datos (Data Path). Consta de dos secciones:

1º Unidad Lógico-Aritmética (ULA o ALU)

Dedicada a efectuar las operaciones lógicas y aritméticas (suma, resta, desplazamiento, rotación, AND, EOR, NO, etc.).

La Máquina Sencilla sólo realiza dos operaciones: SUMA y EOR.

2º Banco de Registros

Lo constituye un grupo de registros en los que se almacenan los datos que actúan como operandos o resultados de las operaciones encomendadas por las instrucciones.

En el caso especial de la Máquina Sencilla, no se dispone de dicho Banco de Registros para mayor simplicidad, lo que implica que todos los datos residen en la Memoria.

La UCP recibe el nombre de **microprocesador** cuando se halla implementada en un único chip.

12.2.2. Memoria

Es el bloque encargado de almacenar la información binaria que compone los datos y las instrucciones. Los unos y ceros que conforman los datos e instrucciones ocupan diversas posiciones de memoria y para su acceso hay que facilitar su dirección.

En el caso de la Máquina Sencilla, la Memoria consta de 128 posiciones, cada una de las cuales contiene la información de una instrucción o un dato, ocupando 16 bits. Para direccionar una posición de esta memoria se precisan 7 líneas digitales ($2^7 = 128$) que implementan el **bus de direcciones.** La información que se lee o escribe en la posición direccionada tiene 16 bits y se transfiere a través del bus de datos. En la figura 12-2 se ofrece una visión de la estructura de la memoria de la Máquina Sencilla, que puesto que puede ser leída y escrita, es de tipo RAM.

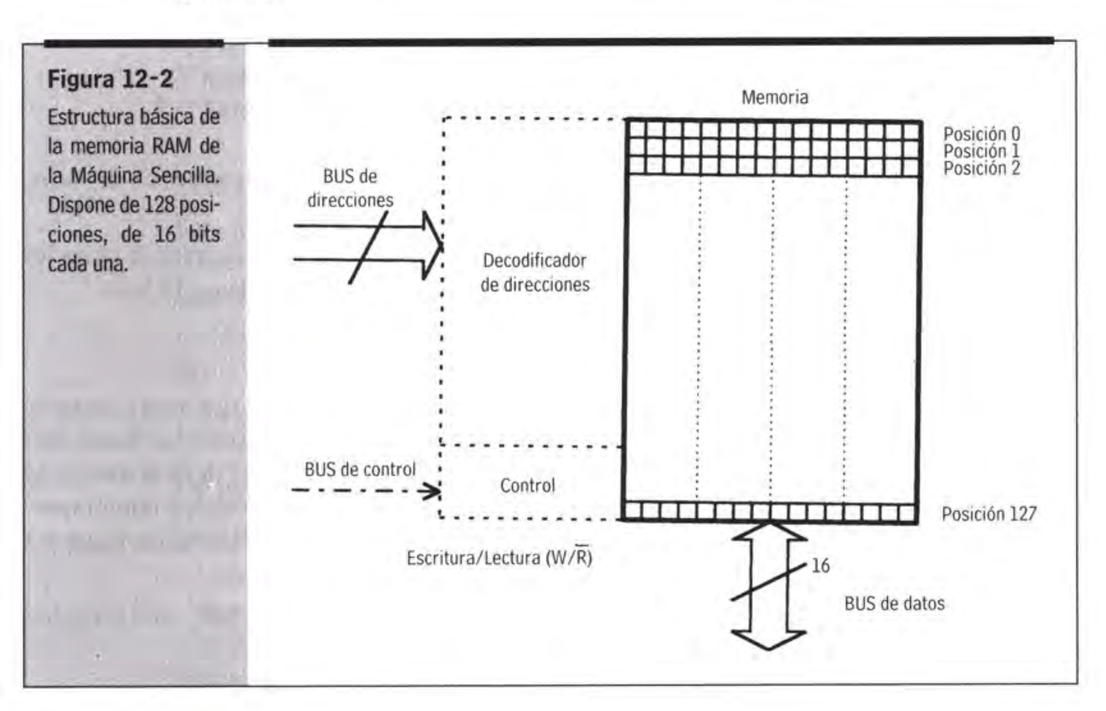


Figura 12-2
Estructura básica de la memoria RAM de la Máquina Sencilla. Dispone de 128 posiciones, de 16 bits cada una.

12.2.3. Módulos de Entrada/Salida

Son los encargados de la comunicación del computador con el exterior a través de diversos periféricos, como teclado, monitor, impresora, etc. De esta forma, el computador se convierte en un sistema abierto, es decir, accesible desde el exterior al tener la posibilidad de poder leer y escribir datos e instrucciones.

En una primera aproximación a la Máquina Sencilla se han eliminado estos módulos de E/S para mayor simplicidad. De esta forma, los resultados obtenidos del procesamiento de los datos por las instrucciones habrá que depositarlos en la Memoria. En consecuencia, en este computador los datos que actúan como operandos de entrada, los datos de resultado u operandos de salida y las instrucciones están almacenados en Memoria.

12.2.4. Los Buses

Son conjuntos de líneas digitales que transfieren información entre los bloques del computador. Hay tres tipos y cada uno de ellos soporta una determinada información.

Bus de Direcciones

La UCP deposita en este conjunto de líneas digitales la información correspondiente a la dirección de la posición de Memoria a la que se desea acceder para leerla o escribir en ella. Se trata de un bus unidireccional, puesto que su información siempre la envía la UCP. En la Máquina Sencilla el bus de direcciones consta de 7 líneas, puesto que la Memoria dispone de 128 posiciones que pueden ser codificadas con 7 bits ($2^7=128$).

Bus de Datos

Es un conjunto de líneas bidireccionales que permite la transferencia de información entre Memoria o E/S y la UCP en los dos sentidos.

En la Máquina Sencilla (MS) el bus de datos tiene 16 líneas, ya que tanto los datos como las instrucciones y las posiciones de Memoria tienen 16 bits.

Bus de Control

Es el grupo de líneas gobernadas por la Unidad de Control, que se encargan de controlar las características de la transferencia a efectuar. Entre las líneas más destacadas de este bus está la de Escritura/Lectura (W/R) que tiene la misión de gobernar el tipo de acceso que se hace a la Memoria. Otra unidad muy importante es la generadora de los impulsos de reloj CLK, que sincroniza todos los componentes.

En la figura 12-3 se muestra un diagrama simplificado de la **MS**, con las especificaciones que se han concretado.

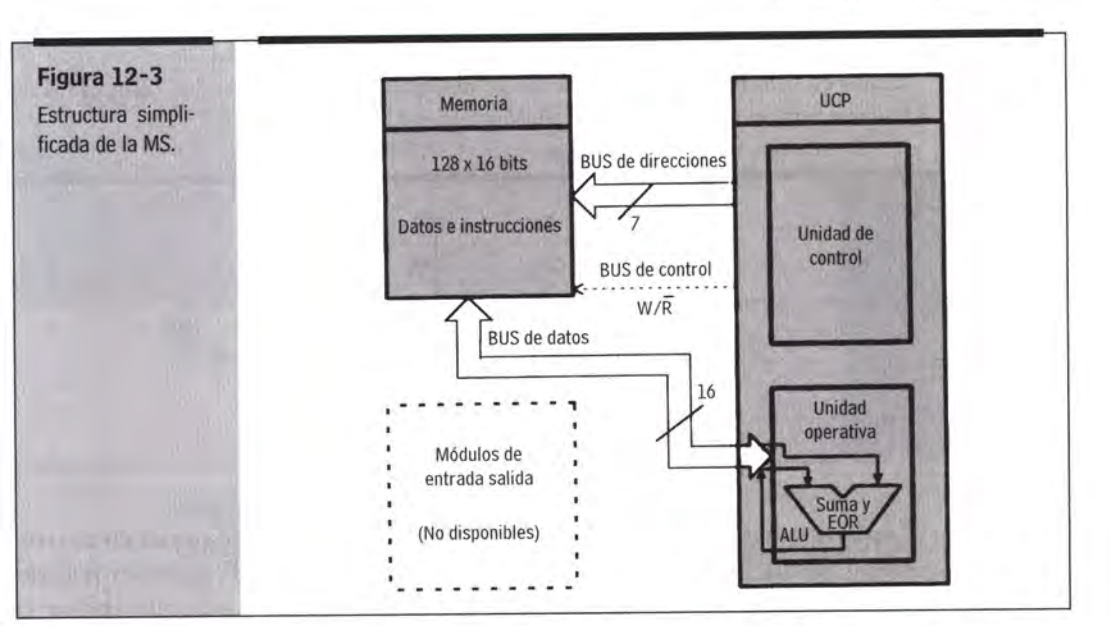


Figura 12-3
Estructura simplificada de la MS.

12.3. DESCRIPCIÓN DE LA MS A NIVEL DE LENGUAJE MÁQUINA

La **MS** va a disponer de una UCP o procesador, capaz de interpretar y ejecutar cuatro instrucciones básicas:

1º **Sumar en Binario: ADD** (las instrucciones se expresan con un nemónico de tres letras procedente de la palabra principal en inglés).
2º **Mover contenidos entre posiciones de Memoria: MOV.**
3º **Comparar contenidos de dos posiciones de Memoria: CMP.**
4º **Saltar si el resultado anterior ha sido cero: BEQ.**

12.3.1. Suma

Esta instrucción sirve para sumar, en binario, el contenido de dos posiciones de la Memoria. Una posición se denomina **Destino: D** y la otra **Fuente: F**. El resultado de la suma se deposita en destino.

La expresión simbólica de esta instrucción usando un **nemónico** que hace recordar la operación expresada en inglés (ADD) es la siguiente:

ADD F, D

De una forma gráfica se puede representar el cometido de esta instrucción de la siguiente manera.

$$(D) \leftarrow (D) + (F)$$

Los símbolos (D) y (F) significan los contenidos binarios de las posiciones de Memoria que ocupan las direcciones D y F.

Para realizar la instrucción ADD, la **MS** dispone de una ALU que es capaz de sumar en binario dos operandos de 16 bits que corresponden a los contenidos de las posiciones de Memoria D y F. Ver figura 12-4.

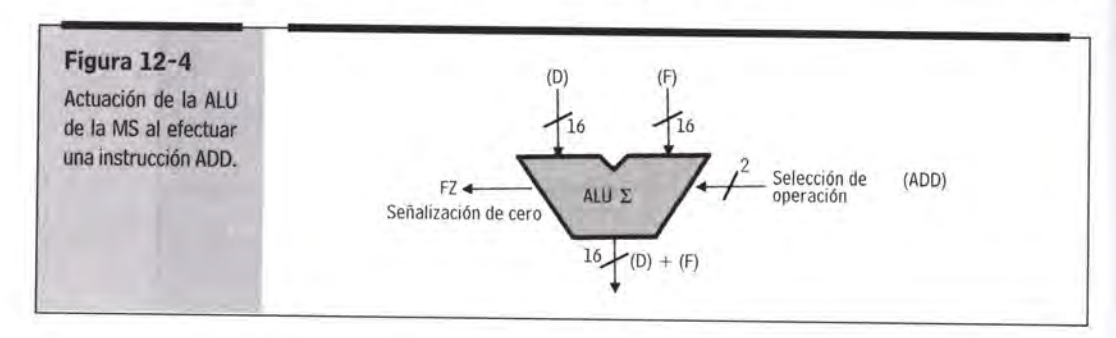


Figura 12-4
Actuación de la ALU de la MS al efectuar una instrucción ADD.

Como se muestra en la figura 12-4, la ALU de la MS puede sumar en binario los 16 bits de la posición D con los 16 bits de la posición F. Al poder realizar cuatro operaciones diferentes, para seleccionar la suma hay que aplicar el código correspondiente a las dos líneas Selección Operación. La salida de la ALU proporciona el resultado de la suma de los dos operandos de 16 bits, entregando un valor de 16 bits. Obsérvese que no dispone de un señalizador que avise si hay acarreo en los bits de más peso; sin embargo, sí dispone de un señalizador (FZ) que avisa cuando el resultado es nulo, o sea, los 16 bits son ceros.

FZ = 1, si (D) + (F) = 0

FZ = 0, si (D) + (F) <>O

12.3.2. Mover

Esta instrucción transfiere el contenido de una posición de Memoria, llamada fuente (F) a otra llamada destino (D). La expresión de esta instrucción empleando el nemónico correspondiente es:

MOV F, D

La representación de la operación de la instrucción MOV F, D es:

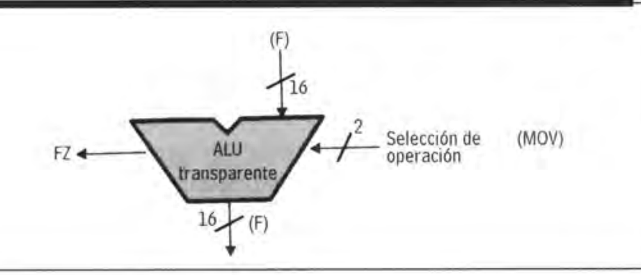
(D) ← (F),

que significa que el contenido (16 bits) de la posición de la Memoria que ocupa la dirección F se transfiere a la posición de Memoria D. El valor previo en D es machacado por el valor de F que se carga allí, mientras que el contenido de la posición de Memoria F no se altera tras la ejecución de la instrucción.

También en esta instrucción actúa el señalizador FZ, al llevarse a cabo la transferencia atravesando la ALU de forma transparente. Figura 12-5.

Figura 12-5

Con la instrucción MOV la ALU deja pasar sin alterar el valor (F). Si (F) = 0 se activa FZ.



El comportamiento de FZ con MOV es:

FZ = 1, si (F) = 0

FZ = 0, si (F) <> 0

12.3.3. Comparar

Compara el contenido de dos posiciones de Memoria, F y D. Esto significa que realiza una operación similar a la resta de ambos contenidos (F)-(D), pero no se obtiene el resultado de dicha sustracción. Sólo se modifica el valor de FZ, que indica si los contenidos comparados son iguales o distintos.

CMP F, D (F) - (D) Sin Resultado

FZ = 1, si (F) - (D) = 0

FZ = 0, si (F) - (D) <> 0

12.3.4. Salto Condicional

Al ejecutarse esta instrucción se produce un salto en el programa según el valor de FZ. Si el señalizador o flag FZ = 1 se pasa a ejecutar la instrucción cuya dirección D se indica detrás del nemónico. En caso contrario, si FZ = 0, se continúa normalmente ejecutando la instrucción siguiente a la del salto.

Debe tenerse en cuenta que la UCP normalmente va ejecutando las instrucciones del programa una detrás de otra, por eso deben estar ordenadas secuencialmente.

Con la instrucción de salto condicional se posibilita la rotura de la secuencia y así saltar a ejecutar una instrucción en cualquier parte del programa. Recibe el nombre de **salto condicional** porque para romper la secuencia se exige que se cumpla una condición, que en este caso es que FZ= 1. Caso de que FZ = 0 se sigue la ejecución secuencial del programa.

El recurso del salto condicional y la rotura de secuencia fue propuesto por **Von Neumann** y es una herramienta potentísima para evitar que la máquina repita siempre el mismo proceso, sin posibilidad de alterarlo según las condiciones particulares de cada caso.

El nemónico de esta instrucción es **BEQ: Branch Equal**, que podría traducirse como Salto si es igual a cero (FZ = 1) .

BEQ actúa sobre un registro contador del que disponen todas las UCP y que se denomina **Contador de Programa, PC**. Su misión es la de proporcionar la direc-

ción de la siguiente instrucción que debe ejecutar el computador. Con la ejecución de todas las instrucciones, excepto con BEQ, el PC se incrementa una unidad para apuntar la dirección siguiente. Con BEQ se comienza explorando FZ; si FZ = 0 se incrementa **PC** como en las demás instrucciones, pero si FZ = 1 la dirección de Memoria donde está la instrucción que a continuación se ejecutará viene especificada por el operando D que acompaña al nemónico y el PC se carga con D+ 1:

$$\text{BEQ D, si FZ} = 1 \quad \text{PC} \leftarrow \text{D} + 1$$
$$\text{BEQ D, si FZ} = 0 \quad \text{PC} \leftarrow \text{PC} + 1$$

Aunque un repertorio de instrucciones tan reducido como el de la MS podría hacer suponer ciertas restricciones, en realidad sus posibilidades sólo están limitadas por la imaginación del programador.

Ejemplo 12-1

Dadas dos variables **a** y **b** residentes en Memoria, confeccionar un programa para la **MS** que calcule su producto y lo almacene en la posición **c**.

SOLUCIÓN

En primer lugar se resuelve el programa empleando un lenguaje de alto nivel, en el cual se ha usado la variable **i** para llevar la cuenta del número de veces que se repite el bucle del programa y que efectúa la suma del multiplicando las veces que indique el multiplicador.

```
begin
        c := 0
        i :=0
        while i < b do
        begin
                c := c + a
                i := i + 1
        end
end
```

El programa, utilizando las instrucciones máquina de la MS es el siguiente:

N.º de Ins.	Etiquetas	Nemónicos	Operandos	Comentarios
1	begin:	MOV	0, c	; c := 0
2		MOV	0, i	; i := 0
3	while:	CMP	i, b	; mientras i < b
4		BEQ	end	; si FZ = 1 salta a "end"
5		ADD	a, c	; c :=c + a
6		ADD	1, i	; i := i + 1
7		CMP	x, x	; siempre FZ = 1
8		BEQ	while	; salto siempre a "while"
9	end:			

Obsérvese que con las dos últimas instrucciones del programa se consigue un salto incondicional, porque al comparar entre sí el contenido de la misma posición de memoria (CMP X, X) siempre FZ = 1, luego la instrucción siguiente BEQ while siempre produce el salto a la posición indicada con la etiqueta while.

12.4. ESTRUCTURA Y MANIPULACIÓN DE LA MEMORIA DE LA MS

La Memoria es el lugar en donde residen las instrucciones y los datos. Sean éstos variables o constantes.

La Memoria de la MS se considera que es de lectura y escritura, tipo RAM, con una capacidad de 128 palabras de 16 bits. Esto significa que el número máximo de las instrucciones más los datos no debe sobrepasar la cantidad de 128.

Tomando como referencia el ejemplo de multiplicación de las variables **a** y **b**, podría repartirse el mapa de memoria de la siguiente forma:

- **Zona de Instrucciones**: comprendida entre las posiciones correspondientes a las direcciones 0 y 8, donde se ubicarán las 9 instrucciones del programa.
- **Zona de Datos Variables**: comprendida entre las direcciones 100 y 103, para las variables **a, b, c** e **i**, respectivamente.
- **Zona de Datos Constantes**: destinada a contener los dos datos constantes, el 1 y el 0, y que están contenidos en las posiciones de memoria de direcciones 104 y 105, respectivamente.
- **Zona de Etiquetas**: en el programa, cuando se efectúa un salto a una instrucción se antepone a esta última una etiqueta que referencia o expresa la dirección de memoria.

En el ejemplo de la multiplicación la etiqueta **begin** corresponde a la instrucción que ocupa la dirección 0, **while** a la que ocupa la dirección 1 y **end** a la que ocupa la dirección 8 de la memoria.

Para distinguir los valores que corresponden a las direcciones de la memoria se les antepone el símbolo @.

En la figura 12-6 se muestra la distribución de la memoria de la **MS** para las instrucciones y los datos del programa de multiplicar.

Figura 12-6
Distribución de las instrucciones y los datos del programa de multiplicar en la Memoria de la MS.

Dirección	Contenido (en binario)	Dirección	Contenido (en binario)
@0	begin: MOV 0, c	@100	a
@1	mov 0, i	@101	b
@2	while: CMP i, b	@102	c
@3	BEQ end	@103	i
@4	ADD a, c	@104	1
@5	ADD 1, i	@105	0
@6	CMP x, x	@106	-----
@7	BEQ while	@107	-----
@8	end:		
@9	-----	@127	-----

12.5. FORMATO BINARIO DE LAS INSTRUCCIONES

Como el computador es una máquina digital y las instrucciones han de almacenarse en la Memoria, éstas se deben codificar en binario. Además, como cada posición de Memoria de la **MS** consta de 16 bits, cada instrucción está compuesta por dicho número de bits, los cuales se organizan en 3 campos:

Campo del Código de Operación (CO)

Consta de 2 bits y sirve para especificar una de las cuatro instrucciones que la **MS** es capaz de ejecutar.

Campo de Operando Fuente (@ Operando Fuente)

Este campo se compone de 7 bits que sirven para determinar la dirección del Operando Fuente que se utiliza en la instrucción.

Se llama **Modo de Direccionamiento** a la forma en la que se expresan en las instrucciones la ubicación de los datos que actúan como operandos. En la **MS** todos los datos están en la Memoria y por lo tanto, para su localización basta indicar la dirección en la que se encuentran. A este modo de Direccionamiento se le denomina **DIRECTO**.

Campo del Operando Destino o Resultado (@ Operando Destino)

Indica la dirección de la Memoria donde se grabará el resultado de la instrucción. En la **MS**, como en otros procesadores, es frecuente que una de las posiciones de la Memoria que actúa como operando fuente también sirva para almacenar el operando destino. Este campo también dispondrá de 7 bits para expresar la dirección del operando destino.

En la figura 12-7 se muestra el formato binario de una instrucción de la **MS**, con los 3 campos descritos.

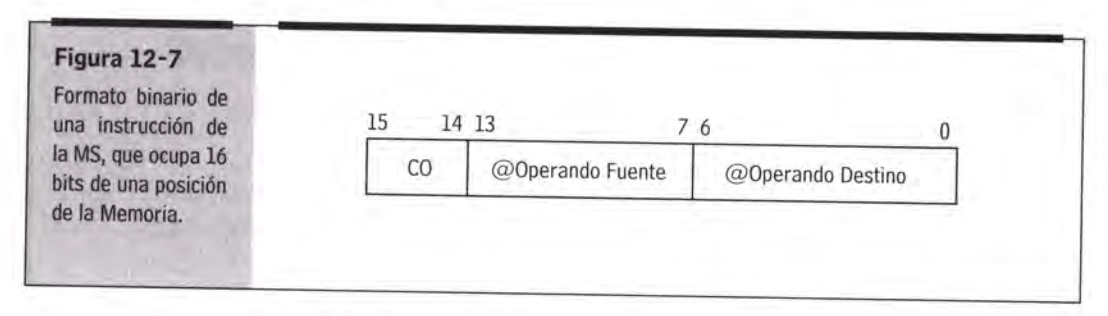


Figura 12-7
Formato binario de una instrucción de la MS, que ocupa 16 bits de una posición de la Memoria.

Con los dos bits del campo CO del Código de Operación se codifican las 4 instrucciones de la MS, de acuerdo con la asignación mostrada en la figura 12-8.

Figura 12-8
Asignación de valores al Código de Operación CO en decimal y binario para las 4 instrucciones de la MS.

CO Decimal	CO Binario	Instrucción
0	00	ADD
1	01	CMP
2	10	MOV
3	11	BEQ

Las instrucciones ADD, CMP y MOV responden exactamente al formato de la figura 12-7, pues las tres disponen de 2 operandos. Se ha elegido una instrucción del programa de multiplicar que ocupe las posiciones de Memoria indicadas en la figura 12-6 para codificarlas en decimal y binario. Ver figura 12-9.

Figura 12-9
Codificación en binario de la instrucción MOV 0, c del programa de multiplicar.

MOV 0, c

CO	@Op. F.	@Op. D.
2	@105	@102

FORMATO EN DECIMAL

CO	@Op. F.	@Op. D.
10	110 1001	110 0110

FORMATO EN BINARIO

La instrucción BEQ sólo dispone de un operando, que es el destino y que corresponde a la dirección de la Memoria usada para apuntar la instrucción a ejecutar a continuación. Para esta instrucción los 7 bits destinados a la dirección del operando fuente son indiferentes y se les designa con una X. Figura 12-10.

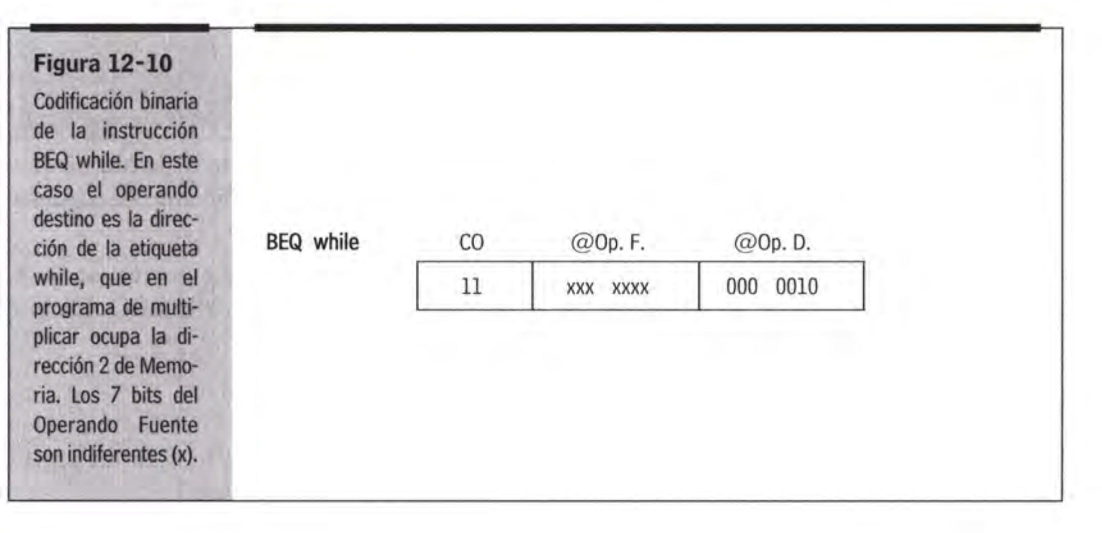


Figura 12-10
Codificación binaria de la instrucción BEQ while. En este caso el operando destino es la dirección de la etiqueta while, que en el programa de multiplicar ocupa la dirección 2 de Memoria. Los 7 bits del Operando Fuente son indiferentes (x).

BEQ while

CO	@Op. F.	@Op. D.
11	xxx xxxx	000 0010

De acuerdo con el formato de la instrucción y las direcciones que ocupan los datos, se presenta en la figura 12-11 el contenido de las diversas posiciones de la Memoria que soportan el programa de multiplicar. En realidad, la Memoria soporta dichos contenidos expresados en binario.

Figura 12-11

Contenido de las posiciones de Memoria con el programa de multiplicar, expresadas en decimal.

Dirección	CO	@Op. Fuente	@Op. Destino	Nemónico
0	2	105	102	MOv 0, c
1	2	105	103	MOV 0, i
2	1	103	101	CMP i, b
3	3	x	008	BEQ end
4	0	100	102	ADD a, c
5	0	104	103	ADD 1, i
6	1	105	105	CMP X, X
7	3	X	002	BEQ while

105	a
101	b
102	c
103	i
104	1
105	0

12.6. LA UNIDAD DE PROCESO

La Unidad de Proceso de un computador, también llamada **Camino de Datos**, es la encargada de llevar a cabo las operaciones lógicas y aritméticas que implican las instrucciones máquina.

Normalmente consta de una Unidad Lógica-Aritmética (ALU) y un Banco de Registros que contiene operandos de acceso rápido. En el caso de la **MS** no dispone de Banco de Registros para utilización del usuario, sino sólo de algunos internos que son transparentes para el programador.

La ALU de la **MS** debe ser capaz de realizar las operaciones que precisan las 4 instrucciones del repertorio de la máquina: MOV, ADD, CMP, BEQ. Dichas operaciones son las siguientes.

1ª Sumar

El circuito encargado de realizar esta operación aritmética se basa en un sumador binario capaz de sumar 2 números de 16 bits. Figura 12-12.

CADIZ

Obsérvese en la figura 12-12 que no se considera ni el acarreo de entrada ni el de salida, lo cual es una limitación de la **MS**.

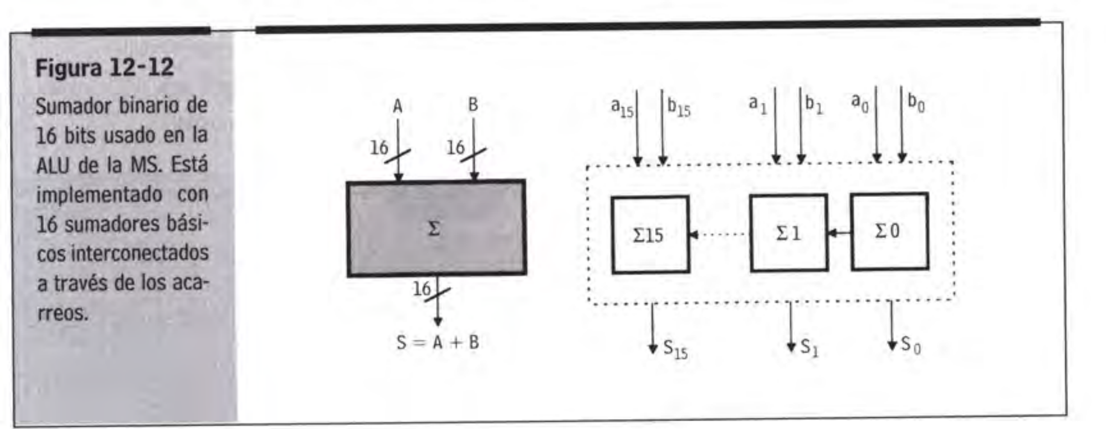


Figura 12-12
Sumador binario de 16 bits usado en la ALU de la MS. Está implementado con 16 sumadores básicos interconectados a través de los acarreos.

2ª Comparar

En este caso la comparación de 2 valores de 16 bits se realiza con 16 puertas EOR, cada una de las cuales compara una pareja de bits. Figura 12-13.

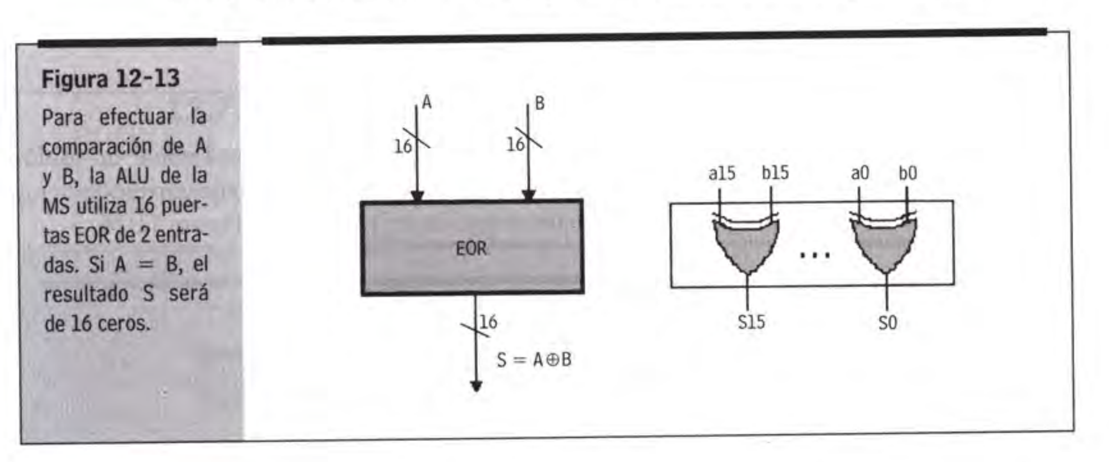


Figura 12-13
Para efectuar la comparación de A y B, la ALU de la MS utiliza 16 puertas EOR de 2 entradas. Si A = B, el resultado S será de 16 ceros.

3ª Ser Transparente

La ALU, para implementar la instrucción MOV, debe ser transparente para uno de sus operandos, es decir, lo debe dejar pasar sin modificarlo. En el caso de la **MS** es el operando B.

Los operandos de entrada A y B de la ALU se almacenan temporalmente en dos registros internos de 16 bits, que llevan sus mismos nombres. En la ALU de la figura 12-14 también se aprecian 2 líneas cuyo código selecciona cualquiera de las 3 posibles operaciones, dejando una combinación sin definir (SELECCIÓN DE OPERACIÓN).

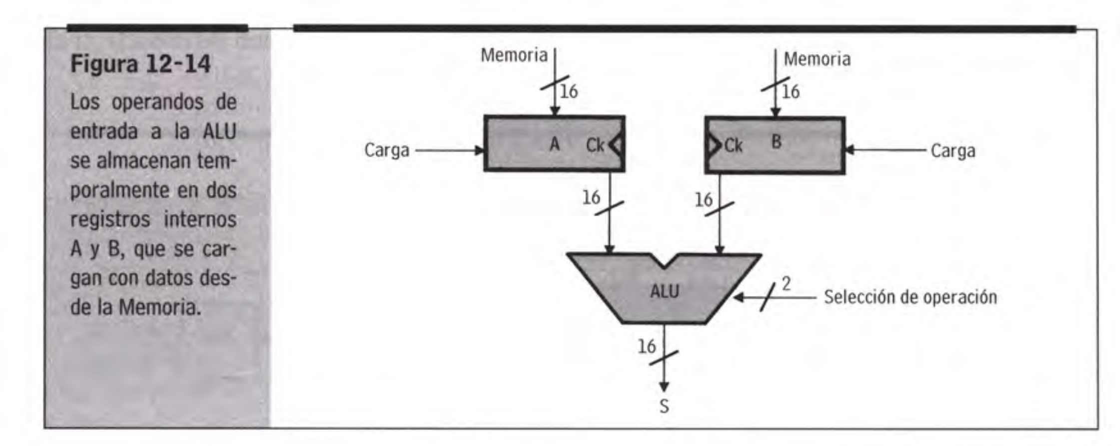


Figura 12-14
Los operandos de entrada a la ALU se almacenan temporalmente en dos registros internos A y B, que se cargan con datos desde la Memoria.

Las dos líneas que seleccionan la operación a realizar por la ALU y que se denominan ALU1 y ALU0 adoptan los siguientes valores para las operaciones correspondientes. Figura 12-15.

Figura 12-15
Codificación de las líneas que seleccionan las operaciones de la ALU.

ALU1	ALU0	OPERACIÓN
0	0	A+B
0	1	A ⊕ B
1	0	B
1	1	No Usada

Teniendo en cuenta los códigos de la figura 12-15, se puede diseñar un multiplexor MPX de 4 entradas que deje pasar a su salida la entrada correspondiente al código ALU1-ALU0, que actúa como control del MPX. Figura 12-16.

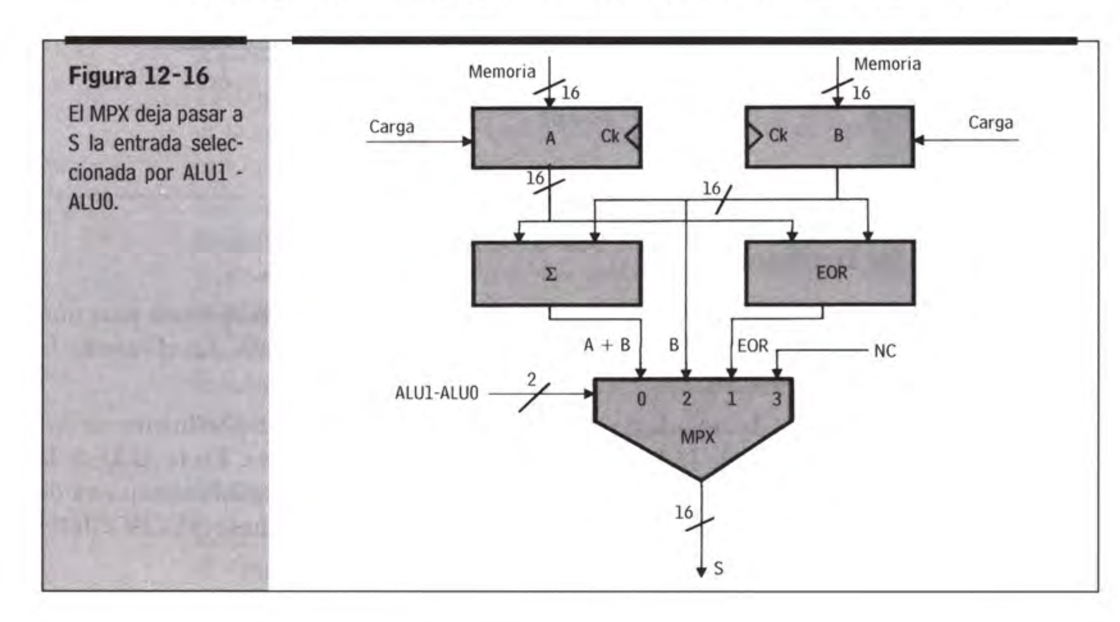


Figura 12-16
El MPX deja pasar a S la entrada seleccionada por ALU1 - ALU0.

El circuito de la figura 12-16 responde a la tabla de codificación de las señales ALU1-ALU0 para la selección de la operación. Así, si ALU1 = ALU0 = 0 por S se obtiene A + B.

El señalizador de cero o flag cero, denominado **FZ**, debe tomar el valor 1 cuando las 16 líneas digitales de la salida de la ALU sean cero. El circuito que detecta 16 líneas a cero es una puerta NOR de 16 entradas, como la de la figura 12-17.

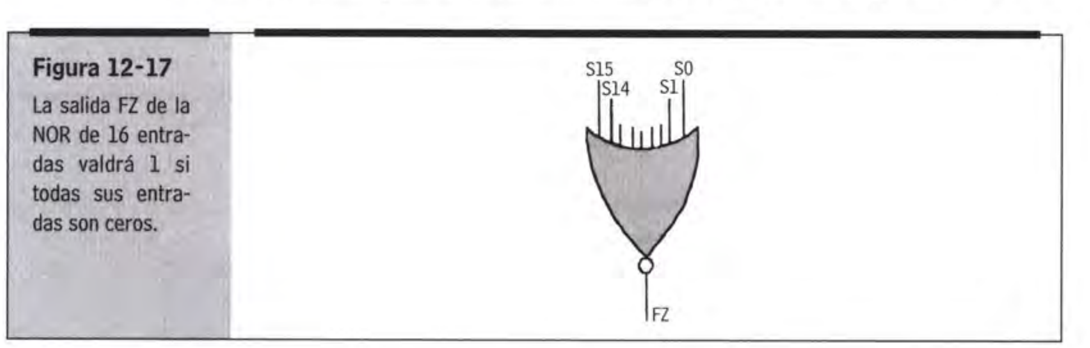


Figura 12-17
La salida FZ de la NOR de 16 entradas valdrá 1 si todas sus entradas son ceros.

Como en el mercado no se comercializan puertas NOR de 16 entradas se implementa el circuito de la figura 12-16, con puertas OR y NOR de 2 entradas, tal como se muestra en la figura 12-18. La salida FZ se almacena en un flip-flop D para mantener su valor hasta que proceda cargar uno nuevo.

12.7. ACCESO A LA MEMORIA

La Memoria de la **MS** es de tipo RAM (Lectura/Escritura) y tiene 128 posiciones de 16 bits cada una, con un bus de direcciones de 7 líneas.

Figura 12-18
Circuito usado para implementar el señalizador de resultado cero (FZ).

A la memoria cabe acceder a leer instrucciones y operandos de entrada a la ALU, o bien a escribir los operandos de salida o resultados.

El acceso a las instrucciones, en general, se realiza por medio del Contador de Programa (PC), que es un registro de 7 bits que guarda la dirección de Memoria donde se encuentra el código de la instrucción que hay que ejecutar a continuación. Normalmente, un programa se va recorriendo instrucción a instrucción secuencialmente, en cuyo caso el PC se incrementa cada vez que comienza la ejecución de una instrucción. Figura 12-19.

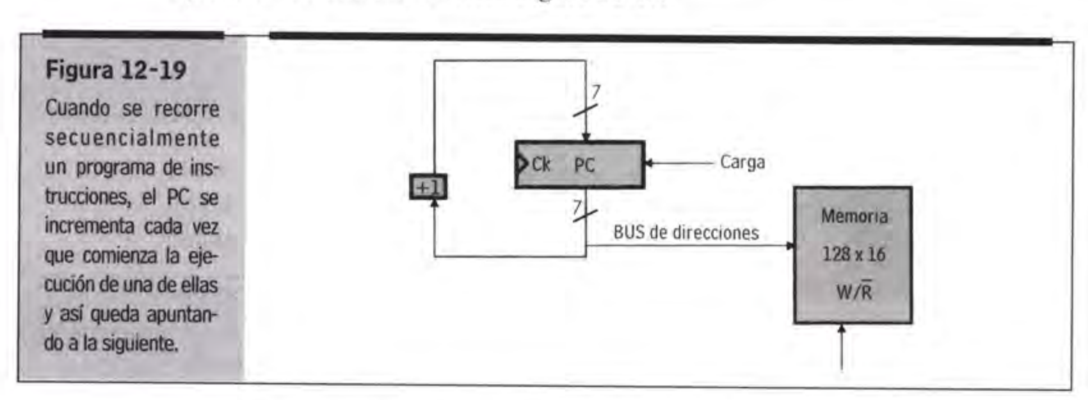


Figura 12-19

Cuando se recorre secuencialmente un programa de instrucciones, el PC se incrementa cada vez que comienza la ejecución de una de ellas y así queda apuntando a la siguiente.

Sin embargo, en las instrucciones BEQ D, si FZ = 1, hay que apuntar a la memoria con el campo D del formato del código de la instrucción para poder realizar una rotura en la secuencia, o sea, un salto.

Por otra parte, cuando hay que acceder a los operandos fuente y destino (F y D), el bus de direcciones debe cargarse con el contenido de dichos campos para direccionar las posiciones de los operandos.

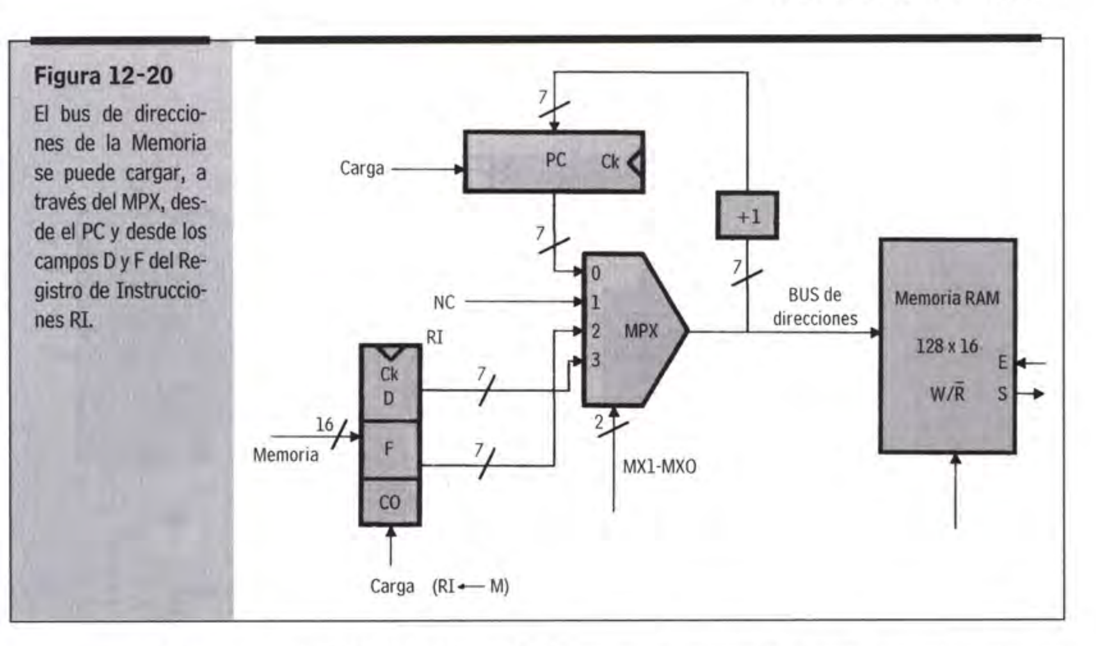
Existe un registro **RI, Registro de Instrucciones**, que guarda los 16 bits del código de cada instrucción, con los campos CO, F y D. RI se carga desde la Memoria cada vez que se lee una instrucción. En la figura 12-20 se muestra un esquema de los diversos elementos capaces de cargar al bus de direcciones de la Memoria. Pueden ser tres: PC, D y F y su selección se realiza con un multiplexor MPX.

En el caso de una instrucción ADD, además de acceder a Memoria para poder cargar en RI el código de dicha instrucción, se precisa acceder otra vez a Memoria para leer el operando fuente (F), después el destino (D) y, por último, tras hacer la suma, otro nuevo acceso a (D) para escribir allí el resultado.

Si se trata de una instrucción BEQ D, tras comprobar que FZ = 1, se debe direccionar la Memoria con el campo D y acceder para recoger el código de la siguiente instrucción a ejecutar. Después la dirección D se incrementa y se carga en el PC para que vuelva a tomar el control del direccionamiento de las instrucciones.

La instrucción CMP F, D, además del acceso a Memoria para recoger el código de la instrucción, luego realiza 2 accesos de lectura a la dirección de los campos F y D. Dichos operandos los carga en los registros A y B para que la ALU realice con ellos la operación EOR. El resultado de la ALU sólo afecta a FZ.

Figura 12-20
El bus de direcciones de la Memoria se puede cargar, a través del MPX, desde el PC y desde los campos D y F del Registro de Instrucciones RI.



La instrucción MOV F, D, lee de Memoria el operando fuente direccionado por F, lo carga en el registro B y la ALU lo deja pasar de forma transparente para escribirlo en la dirección apuntada por el campo D.

En realidad, el control secuencial de todas estas operaciones básicas (microinstrucciones) que se precisan para implementar las instrucciones se va realizando sincronizadamente bajo el gobierno de la Unidad de Control, que se encarga de activar y desactivar en cada momento las señales de control, que gobiernan los diversos registros y MPX de la **MS**.

El código de las señales MPX1 y MPX0 del multiplexor que controla el bus de direcciones se muestra en la figura 12-21.

Figura 12-21
Selección de la dirección de la Memoria mediante el multiplexor y sus señales de control MPX0 y MPX1.

MPX1	MPX0	SELECCIÓN
0	0	PC
0	1	No Conectada
1	0	F
1	1	D

12.8. ESQUEMA DE LA UNIDAD DE PROCESO

En la figura 12-22 se muestra el bloque de la Unidad de Proceso de la **MS** constituido por la ALU y el mecanismo de direccionamiento de la Memoria para acceder a los datos e instrucciones.

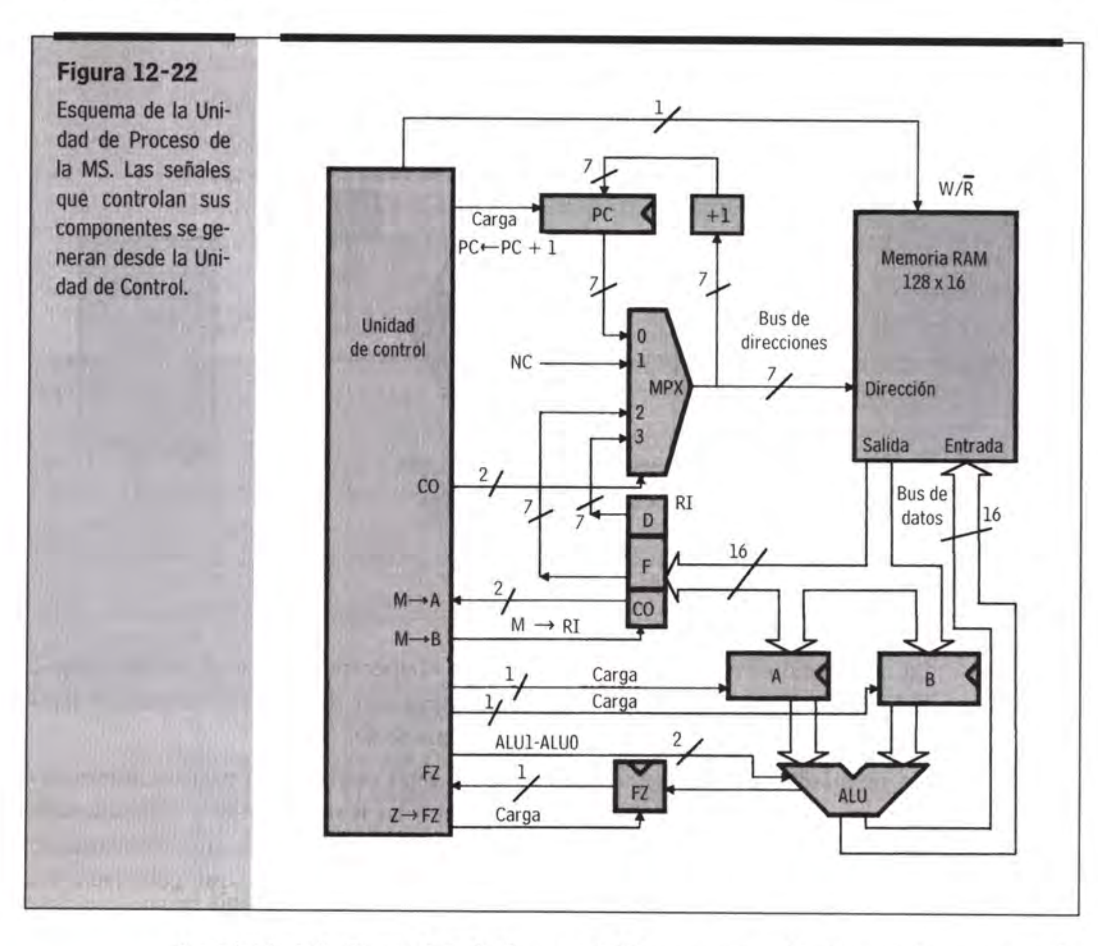


Figura 12-22
Esquema de la Unidad de Proceso de la MS. Las señales que controlan sus componentes se generan desde la Unidad de Control.

La Unidad de Control de la figura 12-22, que se estudiará posteriormente, es la encargada de generar las señales que gobiernan la carga y la selección de los dispositivos de la Unidad de Proceso y de recibir la información del Código de Operación (CO) y del FZ. Se trata de un sistema secuencial de 3 entradas y 10 salidas.

Para estudiar el funcionamiento de la Unidad de Proceso se expresa la activación de las señales que en cada situación se producen representando sus líneas y no dibujando las restantes, que se suponen inactivas.

En toda instrucción, el ciclo comienza accediendo a la Memoria a buscar el código de Operación que se introduce en el RI (Registro de Instrucción). Esta fase común a todas las instrucciones se llama **fase de búsqueda** y puesto que es el PC el que normalmente direcciona la Memoria para acceder al código de la instrucción, se puede expresar:

$$M\ (PC)\ \rightarrow\ RI$$

En la figura 12-23 se presenta un esquema de las señales que se activan y del camino que sigue la información por el bus de direcciones y el bus de datos.

Figura 12-23

Señales que se activan y movimiento de la información por los buses en la fase de búsqueda de una instrucción, que carga en RI el contenido de la posición de Memoria direccionada por PC: M(PC) → RI.

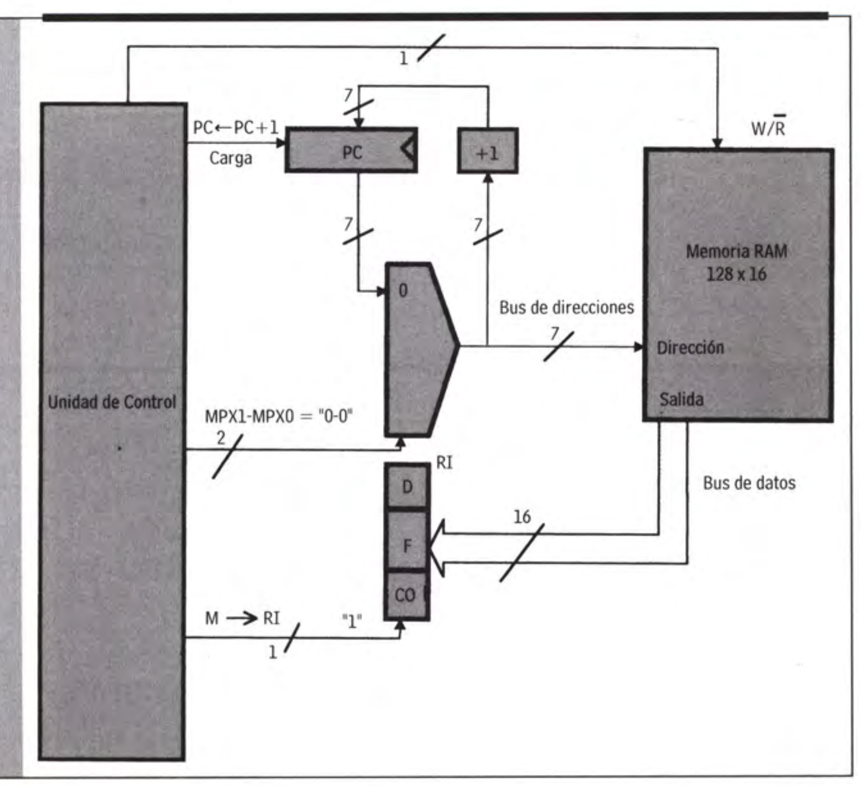
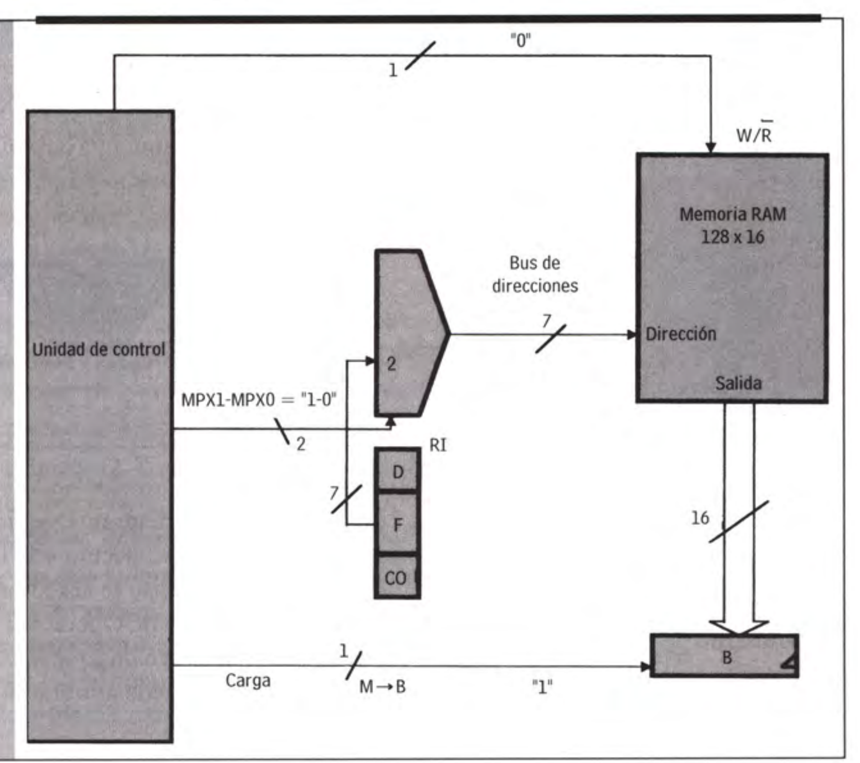


Figura 12-24

Lectura de la posición de Memoria direccionada por el operando fuente F del registro RI y su posterior almacenamiento en el registro B.



En la figura 12-24 se ofrece la representación de las señales que se activan y el flujo de información por los buses para implementar la lectura del operando aportado por el operando fuente (F) del registro RI y su almacenamiento en el registro B de entrada a la ALU.

En la figura 12-25 se presenta el diagrama gráfico de la activación de señales y movimiento de información cuando se trata de almacenar el resultado que sale de la ALU en la posición de Memoria direccionada por el operando destino (D) del registro RI, en una instrucción MOV o ADD. También se carga el Flip-Flop FZ con el valor del flag Z.

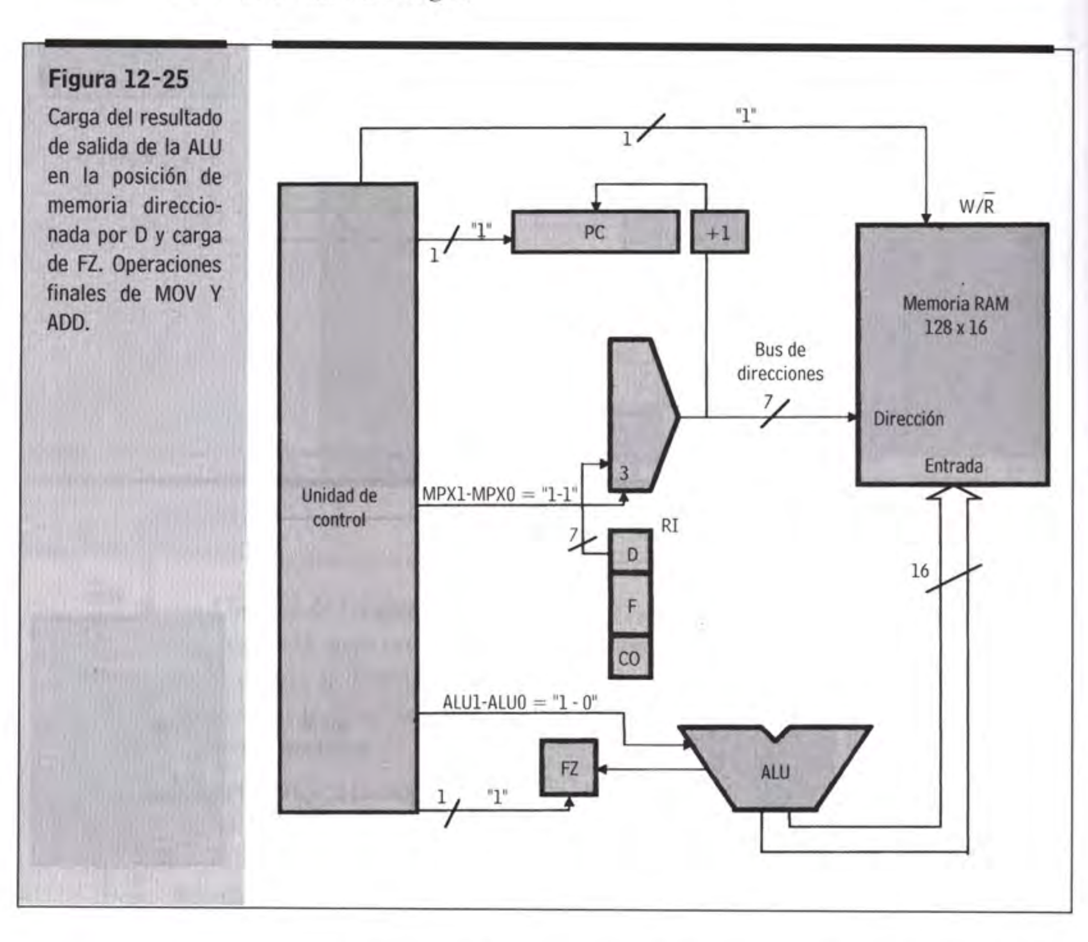


Figura 12-25
Carga del resultado de salida de la ALU en la posición de memoria direccionada por D y carga de FZ. Operaciones finales de MOV Y ADD.

Finalmente, en la figura 12-26 se muestra el diagrama gráfico de la Unidad de Proceso o Camino de Datos de la **MS** cuando se ejecuta una instrucción BEQ D y se produce el salto porque FZ = 1. En este caso se accede a la Memoria con la dirección de D del registro RI y se lee el código de Operación de la siguiente instrucción que se carga en RI. Así mismo, la dirección D se hace pasar por el incrementador +1 y se carga en el PC para que quede apuntando a la siguiente instrucción que hay detrás de la del salto (D+1).

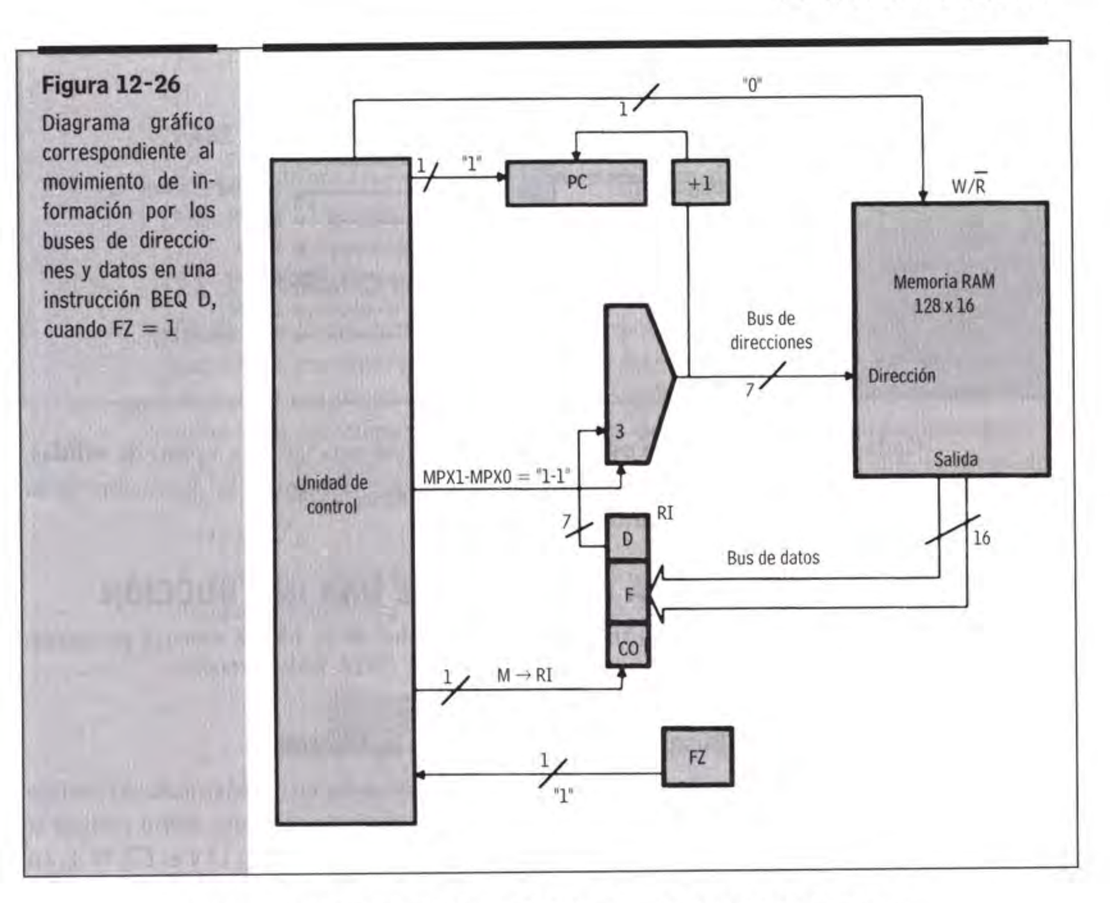


Figura 12-26
Diagrama gráfico correspondiente al movimiento de información por los buses de direcciones y datos en una instrucción BEQ D, cuando FZ = 1

12.9. LA UNIDAD DE CONTROL. GENERALIDADES

La Unidad de Control (UC) es el bloque de la UCP encargado de interpretar el código de Operación de la instrucción en curso y generar todas las señales de control, secuencialmente, para la ejecución de la misma. También explora y tiene en cuenta las señales que condicionan la ejecución de algunas instrucciones, como es el caso de FZ en la BEQ.

Las señales que controlan la actuación de los diversos componentes de la Unidad de Proceso procedían de la Unidad de Control. Así mismo, la Unidad de Proceso proporciona a la Unidad de Control las señales necesarias para la interpretación y ejecución de la instrucción en curso.

En la figura 12-27, se muestra la Unidad de Control de la **MS** recibiendo como señales de entrada las del Código de Operación y FZ y generando como señales de salida las 10 necesarias para gobernar la Unidad de Proceso.

Una vez que la Unidad de Control recibe el Código de Operación CO y el estado de FZ, averigua la instrucción que tiene que ejecutar y la lleva a cabo en varias fases u operaciones elementales, que reciben el nombre de **microinstrucciones**. Cada microinstrucción se efectúa en un ciclo de reloj, y en ella se generan los valores adecuados de las 10 señales binarias de las salidas.

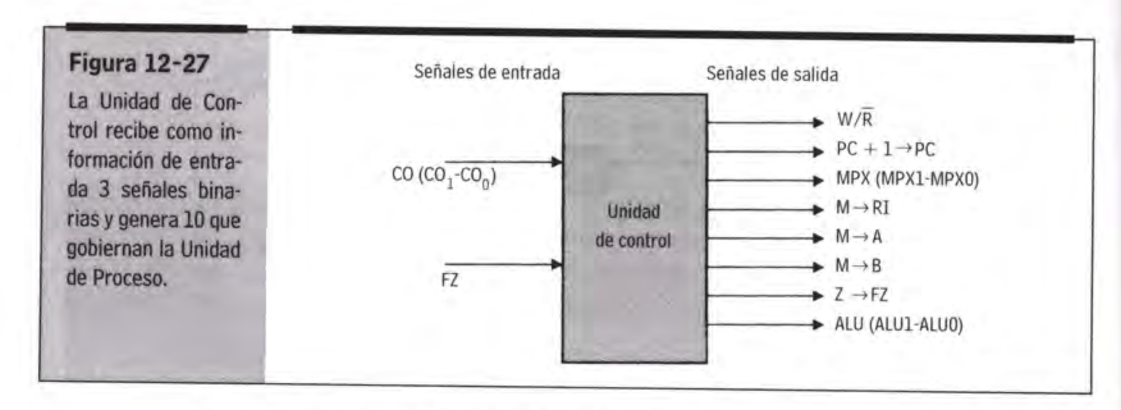


Figura 12-27
La Unidad de Control recibe como información de entrada 3 señales binarias y genera 10 que gobiernan la Unidad de Proceso.

Cada microinstrucción o estado del sistema lleva asociado un **vector de salidas**, que es el valor de las 10 señales de salida que controlan la operación en la Unidad de Proceso o Camino de Datos.

12.10. FASES DE LA EJECUCIÓN DE UNA INSTRUCCIÓN

Cualquiera de las cuatro instrucciones posibles de la **MS** se ejecuta en cuatro fases:

FASE 1: Fase de Búsqueda del Código de la Instrucción

Cualquier instrucción inicia su ejecución buscando en la Memoria su código binario. En general, la dirección de la Memoria que contiene dicho código la proporciona el registro PC, a no ser que se trate de una BEQ D y el FZ = 1, en cuyo caso la dirección es la que expresa el valor D. El contenido de dicha posición se carga en RI y el PC se incrementa una unidad, excepto en la instrucción BEQ D, que se carga con D+1, si FZ= 1.

Gráficamente, esta fase, llamada de búsqueda, puede expresarse así:

M(PC) → RI
PC + 1 → PC

En el caso de una instrucción BEQ D, con FZ = 1:

M(D) → RI
D + 1 → PC

FASE 2: Fase de Decodificación

Se analizan los dos bits que componen el Código de Operación CO y se determina la instrucción a ejecutar.

FASE 3: Búsqueda de Operandos

En esta fase se leen de la Memoria los operandos que van a participar en la instrucción.

En el caso de tratarse de una instrucción BEQ, se evalúa FZ.

FASE 4: Ejecución y Almacenamiento del Resultado

En esta última fase se ejecuta la instrucción y, si procede, se almacena el resultado en la Memoria.

12.11. GRAFO DE ESTADOS

Cada fase de una instrucción se descompone en uno o más estados, que son operaciones elementales denominadas microinstrucciones. Cada estado corresponde a un conjunto de valores de las señales de salida de la Unidad de Control que gobiernan los elementos de la Unidad de Proceso. Las transiciones entre los diversos estados de una instrucción se producen sincrónicamente al ritmo de la señal de reloj y teniendo en cuenta el valor de las tres señales de entrada (CO0-CO1 y FZ).

A continuación se hace una descripción de cada instrucción en función de los diversos estados por los que va pasando.

- Instrucción ADD

Estado	Fase	Operación
0	1	M(PC) → RI; PC+ 1 → PC
1	2	Decodificación de CO0-CO1
2	3	M(F) → B
6	3	M(D) → A
7	4	A+B → M(D); Activar FZ

- Instrucción CMP

Estado	Fase	Operación
0	1	M(PC) → RI; PC+1 → PC
1	2	Decodificación de CO0-CO1
3	3	M(F) → B
8	3	M(D) → A
9	4	A ⊕ B; Activar FZ

- Instrucción MOV

Estado	Fase	Operación
0	1	M(PC) → RI; PC+1 → PC
1	2	Decodificación de CO0-CO1
4	3	M(F) → B
10	4	B → M(D); Activar FZ

- **Instrucción BEQ**

Estado	Fase	Operación
0	1	M(PC) → RI; PC+ 1 → PC
1	2	Evaluación de CO0-CO1
5	3	Evaluación de FZ
11	4	Si FZ = 1; M(D) → RI; D+1 → PC

En el estado 11 de la instrucción BEQ se realiza una fase de búsqueda de la instrucción siguiente, que se halla almacenada en la dirección D, correspondiente al salto condicional y, a continuación, se actualiza el contenido del PC con el valor D + 1.

En la figura 12-28 se muestra un esquema del grafo de estados para cada instrucción de la **MS**. Cada círculo representa el estado (n) y el vector de salida asociado (Sn) que corresponde con los valores que toman las 10 señales digitales de salida de la Unidad de Control para llevar a cabo dicho estado en la Unidad de Proceso. Las transiciones de estados se realizan en función de las tres señales de entrada (CO0-CO1 y FZ) y cuando alguna es indiferente se representa con una X.

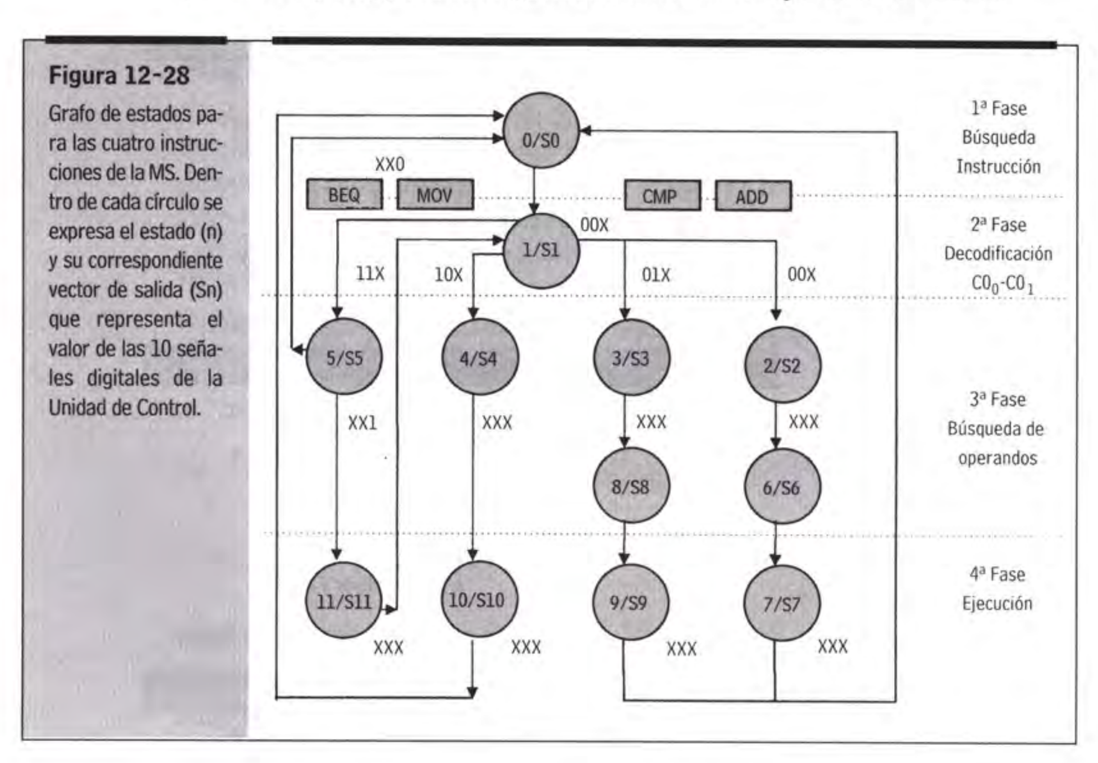


Figura 12-28
Grafo de estados para las cuatro instrucciones de la MS. Dentro de cada círculo se expresa el estado (n) y su correspondiente vector de salida (Sn) que representa el valor de las 10 señales digitales de la Unidad de Control.

En la tabla 12-1 se muestran los valores que toman las 10 señales de salida para cada uno de los 12 estados (S0-S11) que toman las instrucciones en sus fases de ejecución.

Tabla 12-1

Tabla de salidas de la Unidad de Control.

Salida UC	Estados											
--- ---	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11
MPX 1	0	x	1	1	1	x	1	1	1	x	1	1
MPX 0	0	x	0	0	0	x	1	1	1	x	1	1
ALU 1	x	x	x	x	x	x	x	0	x	0	1	x
ALU 0	x	x	x	x	x	x	x	0	x	1	0	x
$W/\overline{R}$	0	0	0	0	0	0	0	1	0	0	1	0
PC+1	1	0	0	0	0	0	0	0	0	0	0	1
M → RI	1	0	0	0	0	0	0	0	0	0	0	1
M → A	0	0	0	0	0	0	1	0	1	0	0	0
M → B	0	0	1	1	1	0	0	0	0	0	0	0
Z → FZ	0	0	0	0	0	0	0	1	0	1	1	0

Se puede comprobar en la tabla anterior que hay varios estados que son idénticos. Si se usa la misma denominación para todos los que son iguales el grafo quedará simplificado. Se comentan las equivalencias:

1.- Las instrucciones ADD, MOV y CMP tienen una fase común que es la carga del operando fuente en el registro D: M(F) → B. Luego los tres estados llamados S2, S3 y S4 pueden reducirse en uno solo que se llamará S2.

2.- Las instrucciones ADD y CMP tienen una forma común, que es la carga del operando destino en el registro A, M(D) → A. Dichos estados S6 y S8 se reducen a uno que será el S6.

3.- La consulta y exploración de FZ en la instrucción BEQ, para decidir si se efectúa el salto, se puede efectuar en el estado S1 y así se evita el estado S5.

Con estas tres simplificaciones el grafo de estados queda reducido al de la figura 12-29.

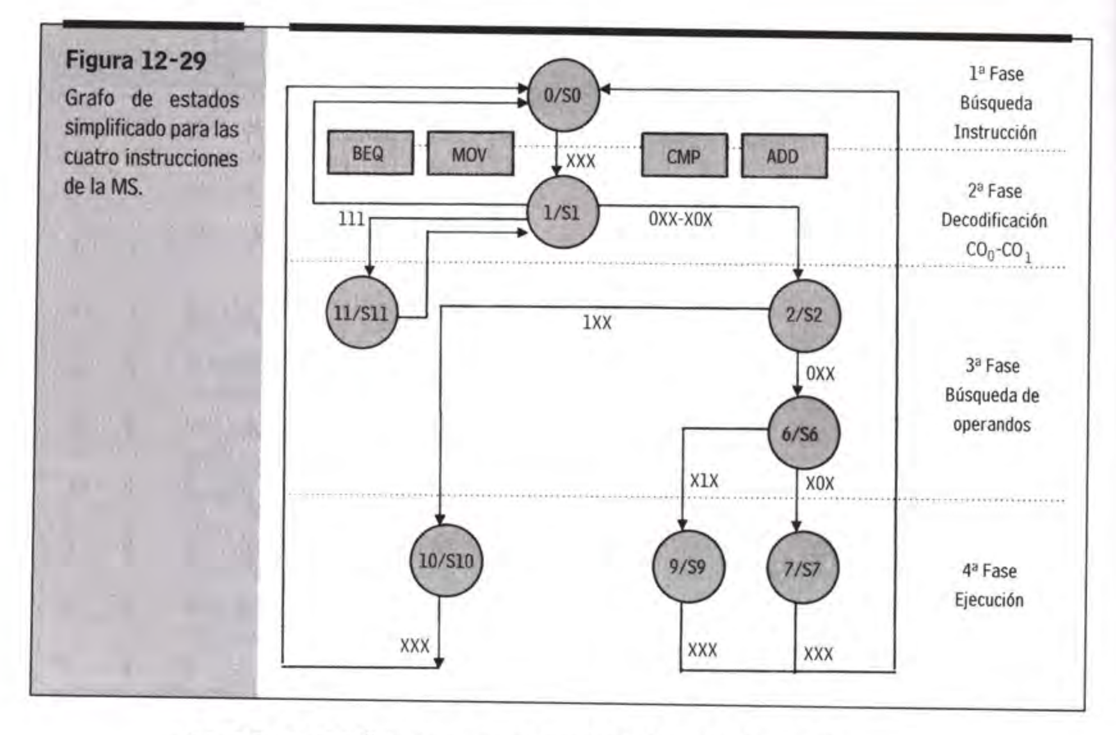


Figura 12-29
Grafo de estados simplificado para las cuatro instrucciones de la MS.

12.12. DISEÑO DE LA UNIDAD DE CONTROL

Con el grafo de estados de la figura 12-29, se conocen los diversos estados y sus correspondientes vectores de salida, en los que se descompone cada instrucción. En total existen 8 estados caracterizados por un conjunto de valores que deben de tomar las 10 señales digitales que genera la Unidad de Control. Por ejemplo, la instrucción ADD consta de 5 estados: S0, S1, S2, S6 y S7; la instrucción BEQ tiene 3 estados (S0, S1 y S11) si FZ = 1, mientras que si FZ = 0 sólo pasa por 2 estados (S0 y S1).

La implementación física de la Unidad de Control se realizará utilizando una memoria no volátil, ROM, de sólo lectura, en la que se grabarán los valores de los vectores correspondientes a cada estado. Al existir 8 estados y cada uno representar un vector con los valores de las 10 señales digitales de salida, sería preciso una memoria ROM de 8 posiciones de 10 bits cada una. Comercialmente se puede encontrar una ROM de 8 x 16, que será la propuesta, aunque no se usarán algunos de sus bits.

Como se aprecia en la tabla 12-2, se asigna a cada dirección de la ROM un estado. La posición de dirección @0 se destina al estado S0, la @1 para S1, etc,... De los 16 bits de cada posición sólo se sacan y se aprovechan los 10 de menos peso, cada uno de los cuales gobierna el estado de una de las señales digitales de salida que parten desde la Unidad de Control hacia la Unidad de Proceso, según la siguiente asignación: b0: Z → FZ, b1: M → B; b2: M → A; b3: M → R1; b4: PC → PC+1; b5: W/$\overline{R}$; b6: ALU0; b7: ALU1; b8: MPX0 y b9: MPX1.

Tabla 12-2
Contenido de la ROM de la Unidad de Control.

Dato/Dirección	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
(S0) @0	x	x	x	x	x	x	0	0	x	x	0	1	1	0	0	0
(S1) @1	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0
(S2) @2	x	x	x	x	x	x	1	0	x	x	0	0	0	0	1	0
(S6) @3	x	x	x	x	x	x	1	1	x	x	0	0	0	1	0	0
(S7) @4	x	x	x	x	x	x	1	1	0	0	1	0	0	0	0	1
(S9) @5	x	x	x	x	x	x	x	x	0	1	0	0	0	0	0	1
(S10) @6	x	x	x	x	x	x	1	1	1	0	1	0	0	0	0	1
(S11) @7	x	x	x	x	x	x	1	1	x	x	0	1	1	0	0	0

Para especificar la dirección de la memoria ROM se precisan 3 líneas llamadas D'0, D'1 y D'2, con las cuales se apunta en cada ciclo de reloj la posición del estado correspondiente a la instrucción en curso.

Para determinar la dirección a leer de la ROM en cada ciclo es necesario tener en cuenta:

1.- El estado anterior, que viene definido por el valor de la dirección de la ROM anterior, que se guardará en 3 flip-flops tipo D (D0, D1 y D2).
2.- El valor de los 2 bits que codifican CO, que se recibe de la Unidad de Proceso.
3.- El valor de FZ.

Se precisa un circuito, denominado **secuenciador,** que reciba como entrada las 6 señales digitales comentadas: D0, D1, D2, CO(2) y FZ y genere en su salida la dirección D'0, D'1 y D'2 correspondiente al siguiente estado de la instrucción que se esté ejecutando.

En la figura 12-30 se muestra el esquema completo de la Unidad de Control, formada por la ROM con los vectores de salida correspondientes a los estados de las instrucciones grabadas, el secuenciador que genera la dirección D'0, D'1 y D'2 del siguiente estado y los 3 flip-flops D que guardan la dirección del estado anterior (D0, D1 y D2).

El diseño del secuenciador sigue las etapas usadas en los circuitos combinacionales:

1.- Tabla de verdad.
2.- Obtención de las ecuaciones lógicas.
3.- Simplificación.
4.- Implementación con puertas lógicas.

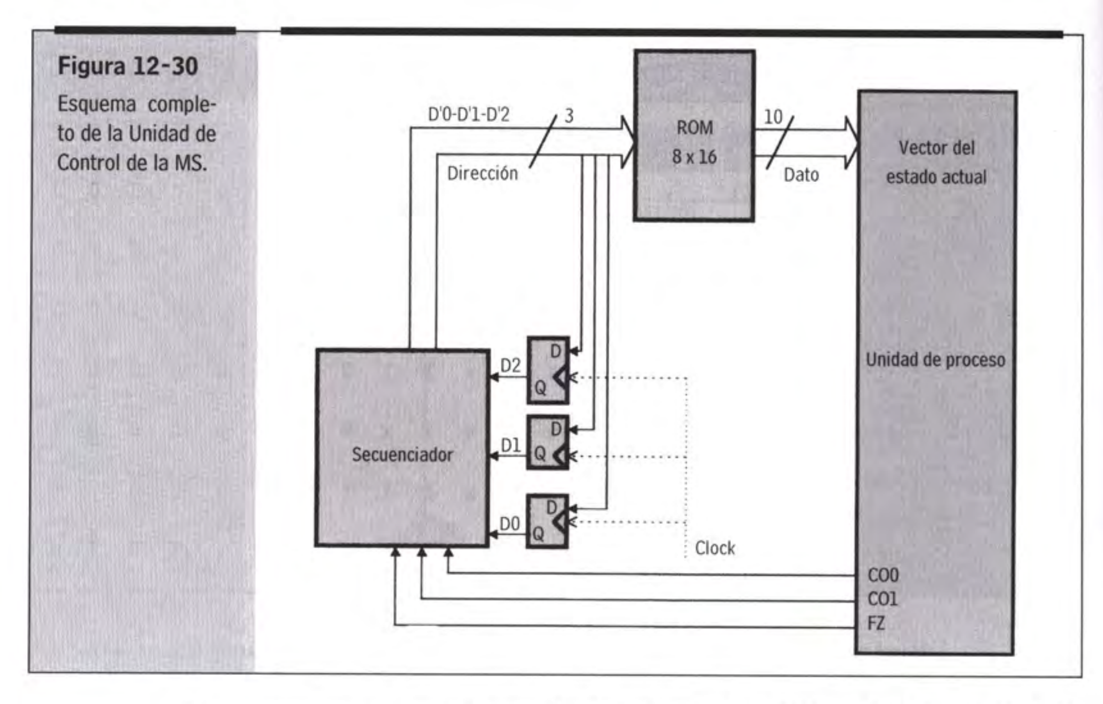


Figura 12-30
Esquema completo de la Unidad de Control de la MS.

De acuerdo con el grafo de estados de la figura 12-29, la tabla de la verdad a la que responden las salidas D'0, D'1 y D'2 es la siguiente (Tabla 12-3):

Tabla 12-3
Tabla de verdad del secuenciador.

Entradas						Salidas		
Estado Presente			Señales de Entrada			Estado Siguiente		
D2	D1	D0	C1	C0	FZ	D'2	D'1	D'0
0	0	0(S0)	X	X	X	0	0	1(S1)
0	0	1(S1)	0	X	X	0	1	0(S2)
			X	0	X	0	1	0(S2)
			1	1	0	0	0	0(S0)
			1	1	1	1	1	1(S11)
0	1	0(S2)	0	X	X	0	1	1(S6)
			1	X	X	1	1	0(S10)
0	1	1(S6)	X	1	X	1	0	1(S9)
			X	0	X	1	0	0(S7)
1	0	0(S7)	X	X	X	0	0	0(S0)
1	0	1(S9)	X	X	X	0	0	0(S0)
1	1	0(S10)	X	X	X	0	0	0(S0)
1	1	1(S11)	X	X	X	0	0	1(S1)

De la tabla de verdad se puede obtener la ecuación de las salidas D'0, D'1 y D'2.

$$D'0 = \overline{D2}\cdot\overline{D1}\cdot\overline{D0} + \overline{D2}\cdot\overline{D1}\cdot D0\cdot C1\cdot C0\cdot FZ + \overline{D2}\cdot D1\cdot\overline{D0}\cdot\overline{C1} + D2\cdot D1\cdot D0\cdot C0 + D2\cdot D1\cdot D0$$

$$D'1 = \overline{D2}\cdot\overline{D1}\cdot D0\cdot\overline{C1} + \overline{D2}\cdot\overline{D1}\cdot D0\cdot\overline{C0} + \overline{D2}\cdot\overline{D1}\cdot D0\cdot C1\cdot C0\cdot FZ + \overline{D2}\cdot D1\cdot\overline{D0}\cdot\overline{C1} + \overline{D2}\cdot D1\cdot\overline{D0}\cdot C1$$

$$D'2 = \overline{D2}\cdot\overline{D1}\cdot D0\cdot C1\cdot C0\cdot FZ + \overline{D2}\cdot D1\cdot\overline{D0}\cdot C1 + \overline{D2}\cdot D1\cdot D0\cdot C0 + \overline{D1}\cdot D1\cdot D0\cdot\overline{C0}$$

Para la implementación física del secuenciador de la Unidad de Control de la **MS**, bastará simplificar las tres ecuaciones y resolverlas utilizando las puertas lógicas adecuadas.

En la figura 12-31 se muestra el esquema lógico que responde en el secuenciador a la ecuación simplificada de la salida D'0.

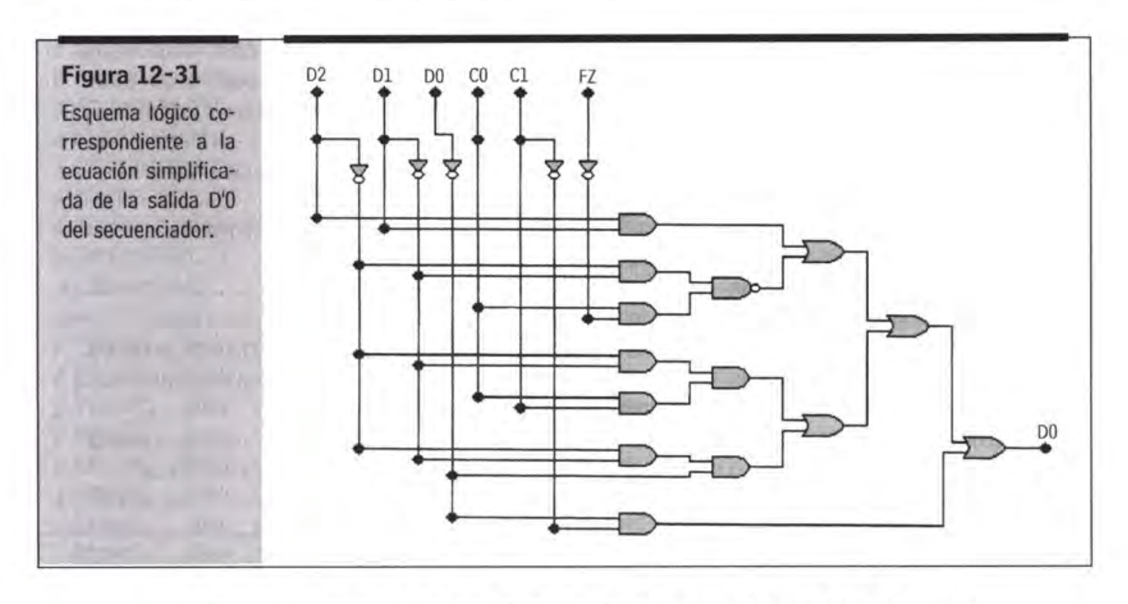


Figura 12-31
Esquema lógico correspondiente a la ecuación simplificada de la salida D'0 del secuenciador.

12.13. EL EMULADOR DE LA MS. INTRODUCCIÓN A LA PROGRAMACIÓN

En el CD que acompaña a este libro se ha incluido un programa muy fácil de manejar que simula el comportamiento de la **MS** y permite observar en la pantalla del computador personal todas las modificaciones que se producen en la Memoria al ejecutar programas. Para su realización, **Ignacio Escalza** ha usado el lenguaje Ensamblador, consiguiendo que su tamaño sea inferior a los 30 Kb. Funciona bajo el Sistema Operativo MS-DOS.

Con este programa de emulación he perseguido acercar al lector a la programación de un computador muy sencillo, empleando el lenguaje máquina.

12.13.1. Cargando el programa

Una vez instalado el programa y situados en el directorio SENCILLA teclearemos: SENCILLA RETURN

y aparecerá la **pantalla 1**:

Pantalla 1

```
+--[Memoria]-----------------------------------------------------------------+
| 00(000)-0B(011)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 0C(012)-17(023)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 18(024)-23(035)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 24(036)-2F(047)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 30(048)-3B(059)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 3C(060)-47(071)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 48(072)-53(083)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 54(084)-5F(095)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 60(096)-6B(107)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 6C(108)-77(119)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 78(120)-7F(127)   0000 0000 0000 0000 0000 0000 0000 0000   Cursor: 00(000)   |
+- Direcciones      Contenidos (Hex) -----------------------------------------+
+---[Programa]-----------------------------------------++----[Registros]----+
| 7B(123)    0000  00 0000000  0000000   ADD 00(000),00(000)   | |                  |
| 7C(124)    0000  00 0000000  0000000   ADD 00(000),00(000)   | | PC=00(000)  FZ=0 |
| 7D(125)    0000  00 0000000  0000000   ADD 00(000),00(000)   | |                  |
| 7E(126)    0000  00 0000000  0000000   ADD 00(000),00(000)   | + - - - - - - - - -+
| 7F(127)    0000  00 0000000  0000000   ADD 00(000),00(000)   | + - - - - - - - - -+
| >00(000)   0000  00 0000000  0000000   ADD 00(000),00(000)   | |  EMULADOR DE LA  |
| 01(001)    0000  00 0000000  0000000   ADD 00(000),00(000)   | | MAQUINA SENCILLA |
| 02(002)    0000  00 0000000  0000000   ADD 00(000),00(000)   | |     (v 2.1)      |
| 03(003)    0000  00 0000000  0000000   ADD 00(000),00(000)   | | F1: Ayuda general|
| 04(004)    0000  00 0000000  0000000   ADD 00(000),00(000)   | | ALT-X: Salir     |
+ Direcc.    Hex.  CO Fuente   Destino   Nemónicos-------------++-----------------+
```

La pantalla se compone de tres ventanas:

a) **Memoria**: 128 posiciones de 16 bits. Se visualizan todos los contenidos de la memoria de la **MS**.

b) **Programa**: instrucciones en Hexadecimal, Binario y desensambladas.

c) **Registros**: PC y flag Z.

12.13.2. Introduciendo datos en la Memoria

La memoria inicialmente está cargada con ceros . Pulsando Alt M nos posicionamos en la ventana de memoria. A la izquierda y en verde tenemos las direcciones (en hexadecimal y entre paréntesis en decimal) y a la derecha el contenido, en amarillo.

La posición actual del cursor aparece resaltada en azul y además viene indicada en la parte inferior derecha de la ventana.

Para introducir un dato en la memoria nos movemos con las flechas del teclado hasta que situamos el cursor sobre la posición. En este ejemplo iremos hasta la dirección 12.

Pulsamos RETURN

Tecleamos el valor:

2 RETURN

Pantalla 2

```
+--[Memoria]------------------------------------------------------------+
| 00(000)-0B(011)  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 0C(012)-17(023)  0002 0003 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 18(024)-23(035)  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 24(036)-2F(047)  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 30(048)-3B(059)  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 3C(060)-47(071)  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 48(072)-53(083)  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 54(084)-5F(095)  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 60(096)-6B(107)  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 6C(108)-77(119)  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 78(120)-7F(127)  0000 0000 0000 0000 0000 0000 0000 0000   Cursor: 0D(013)   |
+- Direcciones     Contenidos (Hex) --------------------------------------+
+---[Programa]-----------------------------------++----[Registros]----+
| 7B(123)   0000  00 0000000  0000000  ADD 00(000),00(000)  ||                    |
| 7C(124)   0000  00 0000000  0000000  ADD 00(000),00(000)  || PC=00(000)   FZ=0  |
| 7D(125)   0000  00 0000000  0000000  ADD 00(000),00(000)  ||                    |
| 7E(126)   0000  00 0000000  0000000  ADD 00(000),00(000)  |+-------------------+
| 7F(127)   0000  00 0000000  0000000  ADD 00(000),00(000)  |+-------------------+
| >00(000)  0000  00 0000000  0000000  ADD 00(000),00(000)  ||  EMULADOR DE LA    |
| 01(001)   0000  00 0000000  0000000  ADD 00(000),00(000)  || MAQUINA SENCILLA   |
| 02(002)   0000  00 0000000  0000000  ADD 00(000),00(000)  ||      (v 2.1)       |
| 03(003)   0000  00 0000000  0000000  ADD 00(000),00(000)  || F1: Ayuda general  |
| 04(004)   0000  00 0000000  0000000  ADD 00(000),00(000)  || ALT-X: Salir       |
+ Direcc.   Hex.  CO Fuente   Destino  Nemónicos--------++----------------+
```

Ahora ya sabemos cómo cargar un valor. Introducimos el dato '3' en la posición 13 de igual manera que antes.

La memoria ha quedado cargada con dos valores que van a ser los operandos de la instrucción que vamos a realizar. Véase la **pantalla 2**.

12.13.3. Introduciendo una instrucción

Con Alt P nos situamos en la ventana de programa. El cursor aparecerá en la posición 0.

La tecla [A] nos permite **ensamblar** a partir de la posición indicada por el cursor. Ensamblar significa que podemos teclear el nemónico y los operandos de una instrucción en la posición de la memoria en que se halle el cursor. El programa se encarga de traducir a código binario y hexadecimal dicha instrucción.

A la derecha, aparecerá una ventana con un cursor en la que escribiremos la instrucción según el formato de la **MS**:

ADD 12, 13 [RETURN]

Para salir del modo 'ensamblar': [Esc]

Ahora tenemos la instrucción cargada en memoria. Si miramos la línea introducida veremos de izquierda a derecha:

1. La dirección de la posición en decimal y hexadecimal en verde: 00(000).
2. El contenido de esa posición, en hexadecimal, en amarillo: 060D (este mismo código estará en la ventana de memoria en la posición 0).
3. A continuación, y de color violeta, el mismo código en binario separado por campos: 00 0001100 0001101.
4. Por último, aparece la instrucción desensamblada con los operandos en decimal y en hexadecimal entre paréntesis. Todo ello de color cyan: ADD 0C(012), 0D(013). Véase la **pantalla 3** correspondiente a este ejemplo.

Pantalla 3

```
+--[Memoria]----------------------------------------------------------+
| 00(000)-0B(011) 060D 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 0C(012)-17(023) 0002 0003 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 18(024)-23(035) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 24(036)-2F(047) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 30(048)-3B(059) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 3C(060)-47(071) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 48(072)-53(083) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 54(084)-5F(095) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 60(096)-6B(107) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 6C(108)-77(119) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 78(120)-7F(127) 0000 0000 0000 0000 0000 0000 0000 0000   Cursor: 0D(013) |
+- Direcciones    Contenidos (Hex) ---------------------------------------+
+---[Programa]----------------------------------++----[Registros]----+
| 7B(123)  0000 00 0000000 0000000 ADD 00(000),00(000) ||                   |
| 7C(124)  0000 00 0000000 0000000 ADD 00(000),00(000) || PC=00(000)  FZ=0 |
| 7D(125)  0000 00 0000000 0000000 ADD 00(000),00(000) ||                   |
| 7E(126)  0000 00 0000000 0000000 ADD 00(000),00(000) |+-------------------+
| 7F(127)  0000 00 0000000 0000000 ADD 00(000),00(000) |+-------------------+
| >00(000) 060D 00 0001100 0001101 ADD 0C(012),0D(013) ||  EMULADOR DE LA   |
| 01(001)  0000 00 0000000 0000000 ADD 00(000),00(000) || MAQUINA SENCILLA  |
| 02(002)  0000 00 0000000 0000000 ADD 00(000),00(000) ||      (v 2.1)      |
| 03(003)  0000 00 0000000 0000000 ADD 00(000),00(000) || F1: Ayuda general |
| 04(004)  0000 00 0000000 0000000 ADD 00(000),00(000) || ALT-X: Salir      |
+ Direcc.  Hex. CO Fuente  Destino Nemónicos--------++------------------+
```

12.13.4. Ejecutando una instrucción

Estando situados en la ventana de programa nos aseguramos de que el PC apunte a la dirección donde está la instrucción. Si no fuese así, nos situaremos en la instrucción mediante las flechas y pulsaremos 'barra espaciadora'.

Para ejecutar la instrucción: T (Tracear).

Ahora, la instrucción se habrá ejecutado y por lo tanto el PC indicará a la siguiente posición (PC = 1). Podemos comprobarlo en la ventana de registros. En la ventana de programa el cursor está sobre la dirección a la que apunta el PC (dirección 1).

En la ventana de memoria vemos que ha habido un solo cambio en la posición 13 (destino de operación). El valor allí contenido es el resultado de la suma, en este caso '5'.

El contenido de la posición fuente (pos. 12) no ha variado. En la ventana de registros vemos que el flag Z no se ha activado (FZ = 0), ya que el resultado no ha sido 'cero'. Se muestra la **pantalla 4**.

Pantalla 4

```
+--[Memoria]-------------------------------------------------------------------+
| 00(000)-0B(011) 060D      0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 0C(012)-17(023) 0002      0005 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 18(024)-23(035)      0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 24(036)-2F(047)      0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 30(048)-3B(059)      0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 3C(060)-47(071)      0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 48(072)-53(083)      0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 54(084)-5F(095)      0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 60(096)-6B(107)      0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 6C(108)-77(119)      0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 78(120)-7F(127)      0000 0000 0000 0000 0000 0000 0000 0000   Cursor: 0D(013)    |
+- Direcciones         Contenidos (Hex) --------------------------------------------+
+---[Programa]------------------------------------------++----[Registros]----+
| 7C(124)   0000  00 0000000  0000000   ADD  00(000),00(000)  ||                   |
| 7D(125)   0000  00 0000000  0000000   ADD  00(000),00(000)  || PC=01(001)   FZ=0 |
| 7E(126)   0000  00 0000000  0000000   ADD  00(000),00(000)  ||                   |
| 7F(127)   0000  00 0000000  0000000   ADD  00(000),00(000)  |+ - - - - - - - - - +
| 00(000)   060D  00 0001100  0001101   ADD  0C(012),0D(013)  |+ - - - - - - - - - +
|>01(001)   0000  00 0000000  0000000   ADD  00(000),00(000)  ||  EMULADOR DE LA   |
| 02(002)   0000  00 0000000  0000000   ADD  00(000),00(000)  || MAQUINA SENCILLA  |
| 03(003)   0000  00 0000000  0000000   ADD  00(000),00(000)  ||      (v 2.1)      |
| 04(004)   0000  00 0000000  0000000   ADD  00(000),00(000)  || F1: Ayuda general |
| 05(005)   0000  00 0000000  0000000   ADD  00(000),00(000)  || ALT-X: Salir      |
+ Direcc.   Hex.  CO Fuente   Destino   Nemónicos-----------++-------------------+
```

12.13.5. Un programa ejemplo

Partiendo de los siguientes valores de memoria:

posición	valor
24	8
25	5
26	D

Cargar el siguiente programa a partir de la posición 0:

00	ADD 24, 25
01	CMP 25, 26
02	BEQ 04
03	MOV 25, 60
04	MOV 24,84

Se introducirán todas las instrucciones, una a una, pulsando Esc tras la última. Una vez cargado el programa, se obtendrá la **pantalla 5**.

Pantalla 5

```
+--[Memoria]-----------------------------------------------------------+
| 00(000)-0B(011) 0C19 4C9A C004 8CBC 8C54 0000 0000 0000 0000 0000 0000 0000 |
| 0C(012)-17(023) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 18(024)-23(035) 0008 0005 000D 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 24(036)-2F(047) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 30(048)-3B(059) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 3C(060)-47(071) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 48(072)-53(083) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 54(084)-5F(095) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 60(096)-6B(107) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 6C(108)-77(119) 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 78(120)-7F(127) 0000 0000 0000 0000 0000 0000 0000 0000   Cursor: 1A(026)   |
+- Direcciones    Contenidos (Hex) ---------------------------------------+
+---[Programa]---------------------------------------++----[Registros]----+
| >00(000) 0C19 00 0011000 0011001 ADD 18(024),19(025) ||                  |
| 01(001)  4C9A 01 0011001 0011010 CMP 19(025),1A(026) || PC=00(000) FZ=0  |
| 02(002)  C004 11 0000000 0000100 BEQ        04(004)  ||                  |
| 03(003)  8CBC 10 0011001 0111100 MOV 19(025),3C(060) |+------------------+
| 04(004)  8C54 10 0011000 1010100 MOV 18(024),54(084) |+------------------+
| 05(005)  0000 00 0000000 0000000 ADD 00(000),00(000) ||  EMULADOR DE LA  |
| 06(006)  0000 00 0000000 0000000 ADD 00(000),00(000) || MAQUINA SENCILLA |
| 07(007)  0000 00 0000000 0000000 ADD 00(000),00(000) ||      (v 2.1)     |
| 08(008)  0000 00 0000000 0000000 ADD 00(000),00(000) || F1: Ayuda general|
| 09(009)  0000 00 0000000 0000000 ADD 00(000),00(000) || ALT-X: Salir     |
+ Direcc.  Hex.  CO Fuente  Destino Nemónicos--------++------------------+
```

Si se desea, puede grabarse el programa en disco para tenerlo guardado y poder recuperarlo más adelante. Para ello, pulsaremos F2. Nos aparecerá una pantalla pidiéndonos el nombre del fichero; lo introduciremos con el teclado y pulsaremos RETURN.

Cuando queramos cargar el fichero, actuaremos de forma análoga, pero pulsando F3 en lugar de F2.

Antes de proceder a la ejecución del programa, haremos apuntar el PC a la dirección 0, para lo cual se lleva el cursor a dicha dirección y se pulsa la barra espaciadora para actualizar el PC.

Ahora, ejecutaremos el programa instrucción a instrucción. Tecla T.

Al completar el programa aparecerá la **pantalla 6**.

Pantalla 6

```
+--[Memoria]-------------------------------------------------------------------+
| 00(000)-0B(011)   0C19 4C9A C004 8CBC 8C54 0000 0000 0000 0000 0000 0000 0000 |
| 0C(012)-17(023)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 18(024)-23(035)   0008 000D 000D 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 24(036)-2F(047)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 30(048)-3B(059)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 3C(060)-47(071)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 48(072)-53(083)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 54(084)-5F(095)   000D 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 60(096)-6B(107)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 6C(108)-77(119)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 78(120)-7F(127)   0000 0000 0000 0000 0000 0000 0000 0000    Cursor: 1A(026)  |
+- Direcciones      Contenidos (Hex) ------------------------------------------+
+---[Programa]-----------------------------------------++----[Registros]----+
| 00(000)   0C19 00 0011000 0011001  ADD 18(024),19(025) ||                   |
| 01(001)   4C9A 01 0011001 0011010  CMP 19(025),1A(026) || PC=05(005)  FZ=0 |
| 02(002)   C004 11 0000000 0000100  BEQ         04(004) ||                   |
| 03(003)   8CBC 10 0011001 0111100  MOV 19(025),3C(060) |+------------------+
| 04(004)   8C54 10 0011000 1010100  MOV 18(024),54(084) |+------------------+
|>05(005)   0000 00 0000000 0000000  ADD 00(000),00(000) ||  EMULADOR DE LA   |
| 06(006)   0000 00 0000000 0000000  ADD 00(000),00(000) || MAQUINA SENCILLA  |
| 07(007)   0000 00 0000000 0000000  ADD 00(000),00(000) ||      (v 2.1)      |
| 08(008)   0000 00 0000000 0000000  ADD 00(000),00(000) || F1: Ayuda general |
| 09(009)   0000 00 0000000 0000000  ADD 00(000),00(000) || ALT-X: Salir      |
+ Direcc.   Hex. CO Fuente  Destino  Nemónicos---------++-------------------+
```

12.13.6. Un programa más complicado

Vamos a construir un breve programa para que la **MS** calcule el producto de dos números 'A y 'B y lo almacene en 'C.

Primero diseñaremos un algoritmo en lenguaje de alto nivel:

```
begin
    C := 0;
    i := 0;
    while i<B do
        begin
        C :=C + A;
        i := i + 1;
    end
end
```

Traduciendo el algoritmo al repertorio de instrucciones de **MS**:

```
begin:  MOV 0, C      ;C := 0
        MOV 0, i      ;I := 0
while:  CMP i, B      ; mientras i<B
        BEQ end
        ADD A, C      ;C := C+A
        CMP x, x      ; FZ := 1
        BEQ while     ; salto incondicional
end:
```

Es importante notar que las dos últimas instrucciones implementan un salto incondicional. Primero, la comparación de una posición consigo misma activa FZ para, a continuación, poderse ejecutar el salto siempre.

El algoritmo precisa 2 datos constantes: el 0 (para poner a 0 las variables i y C) y el 1 (para incrementar i). Dado que la **MS** no admite el direccionamiento inmediato, estos datos deberán estar en memoria. Elegiremos las posiciones 120 y 121 para el 0 y el 1, respectivamente. Además, tenemos 2 constantes (A y B) y dos variables (i y C). Utilizaremos las posiciones 108, 109, 110 y 111 para almacenar A, B, C e i, respectivamente.

Sobre la ventana de Memoria se introduce el 0 y el 1 en 120 y 121 y los datos que queramos multiplicar (por ejemplo 3 y 4) en 108 y 109.

A continuación, introduciremos el programa a partir de la posición 0 y pondremos el PC apuntando a 0 (lo podemos hacer también pulsando Ctrl I e introduciendo un 0, desde la ventana de programa pulsando P e introduciendo un 0 o haciendo un reset de la **MS** pulsando Ctrl F10 que además pone FZ a 0, con lo que aparecerá la **pantalla 7**.

Pantalla 7

```
+--[Memoria]----------------------------------------------------------------+
| 00(000)-0B(011)   BC6E BC6F 77ED C008 366E 3CEF 4000 C002 0000 0000 0000 0000 |
| 0C(012)-17(023)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 18(024)-23(035)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 24(036)-2F(047)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 30(048)-3B(059)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 3C(060)-47(071)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 48(072)-53(083)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 54(084)-5F(095)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 60(096)-6B(107)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 6C(108)-77(119)   0003 0004 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |
| 78(120)-7F(127)   0000 0000 0000 0000 0000 0000 0000 0000   Cursor: 6C(108)   |
+- Direcciones      Contenidos (Hex) -----------------------------------------+
+---[Programa]-----------------------------------------++----[Registros]----+
| >00(000)  BC6E 10 1111000 1101110  MOV 78(120),6E(110)  | |                    |
| 01(001)   BC6F 10 1111000 1101111  MOV 78(120),6F(111)  | | PC=00(000)   FZ=0  |
| 02(002)   77ED 01 1101111 1101101  CMP 6F(111),6D(109)  | |                    |
| 03(003)   C008 11 0000000 0001000  BEQ         08(008)  | +------------------+
| 04(004)   366E 00 1101100 1101110  ADD 6C(108),6E(110)  | +------------------+
| 05(005)   3CEF 00 1111001 1101111  ADD 79(121),6F(111)  | |  EMULADOR DE LA    |
| 06(006)   4000 01 0000000 0000000  CMP 00(000),00(000)  | | MAQUINA SENCILLA   |
| 07(007)   C002 11 0000000 0000010  BEQ         02(002)  | |      (v 2.1)       |
| 08(008)   0000 00 0000000 0000000  ADD 00(000),00(000)  | | F1: Ayuda general  |
| 09(009)   0000 00 0000000 0000000  ADD 00(000),00(000)  | | ALT-X: Salir       |
+ Direcc.   Hex. CO Fuente  Destino  Nemónicos--------++-----------------+
```

Podríamos tracear el programa instrucción a instrucción como antes, pero tardaríamos bastante tiempo. Para solucionarlo vamos a poner un Break Point (Punto de parada) en la dirección 8, que es donde acaba el programa. Situaremos el cursor en dicha posición y pulsaremos RETURN .

Observaremos que la línea cambia de color para indicar que el Break Point está activo.

Para ejecutar, pulsaremos la tecla G. Como podemos ver en la **pantalla 8** siguiente, el programa se ha ejecutado y el resultado de la multiplicación ocupa la posición 110.

También podemos usar la tecla P en lugar de la G. La diferencia es que con la 1ª se irán actualizando las ventanas tras la ejecución de cada instrucción, mientras que con la 2ª solamente lo harán al final.

Probaremos introduciendo otros valores para A y B en las posiciones 108 y 109 y comprobaremos el resultado en 110.

Pantalla 8

```
+--[Memoria]---------------------------------------------------------+
| 00(000)-0B(011)   78DC BC6F 77ED C008 366E 3CEF 4000 C002 0000 0000 0000 0000  |
| 0C(012)-17(023)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  |
| 18(024)-23(035)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  |
| 24(036)-2F(047)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  |
| 30(048)-3B(059)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  |
| 3C(060)-47(071)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  |
| 48(072)-53(083)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  |
| 54(084)-5F(095)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  |
| 60(096)-6B(107)   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  |
| 6C(108)-77(119)   0003 0004 000C 0004 0000 0000 0000 0000 0000 0000 0000 0000  |
| 78(120)-7F(127)   0000 0001 0000 0000 0000 0000 0000 0000   Cursor: 6C(108)    |
+- Direcciones      Contenidos (Hex) ------------------------------------------+
+---[Programa]------------------------------------------++----[Registros]----+
| 03(003)   C008  11 0000000 0001000   BEQ          08(008)  | |                   |
| 04(004)   366E  00 1101100 1101110   ADD  6C(108),6E(110)  | | PC=08(008)   FZ=0 |
| 05(005)   3CEF  00 1111001 1101111   ADD  79(121),6F(111)  | |                   |
| 06(006)   4000  01 0000000 0000000   CMP  00(000),00(000)  | +-----------------+
| 07(007)   C002  11 0000000 0000010   BEQ          02(002)  | +-----------------+
| >08(008)  0000  00 0000000 0000000   ADD  00(000),00(000)  | |  EMULADOR DE LA   |
| 09(009)   0000  00 0000000 0000000   ADD  00(000),00(000)  | | MAQUINA SENCILLA  |
| 0A(010)   0000  00 0000000 0000000   ADD  00(000),00(000)  | |      (v 2.1)      |
| 0B(011)   0000  00 0000000 0000000   ADD  00(000),00(000)  | | F1: Ayuda general |
| 0C(012)   0000  00 0000000 0000000   ADD  00(000),00(000)  | | ALT-X: Salir      |
+ Direcc.   Hex.  CO Fuente  Destino   Nemónicos---------++-----------------+
```

12.13.7. Resumen de órdenes y comandos

El programa funciona con tres ventanas principales:

1. **Ventana de Memoria**: en ella se muestra el contenido de la memoria de la **MS** en hexadecimal. Se pueden modificar los contenidos.
2. **Ventana de Programa**: en esta ventana se muestra el contenido de la memoria, pero desensamblado. Se pueden introducir nemónicos y ejecutarlos.
3. **Ventana de Registros**: en ella se visualiza el registro PC (Contador de Programa) y el Flag Z.

En todo momento hay una ventana activa, que es la que está rodeada por un recuadro blanco de doble trazo. Hay teclas generales que funcionan en todas las ventanas, y teclas de cada ventana, las cuales funcionan sólo en la ventana activa.

Todas las direcciones de memoria (en cualquiera de las ventanas) se especifican en hexadecimal y a continuación se muestra su equivalente decimal entre paréntesis: **HH(DDD)**.

Así mismo, cuando hagamos alusión a direcciones de memoria (en operaciones de bloques, actualizaciones del PC -Contador de Programa- o ensamblado de instrucciones) se podrán especificar tanto en decimal como en hexadecimal. En este último caso, introduciremos la letra 'H al comienzo del número.

El programa se puede ejecutar de tres formas:

1.- Escribiendo **SENCILLA**, sin parámetros.
El programa se ejecutará en el modo de vídeo de 80x25.

2.- Escribiendo **SENCILLA 0**.
El programa se ejecutará en el modo de vídeo actual; si bien no funcionará correctamente con modos que no sean de 80 columnas.

3.- Escribiendo **SENCILLA 1**.
El programa se ejecutará en el modo de vídeo EGA/VGA de 43/50 filas.

TECLAS GENERALES

A continuación, se describen todas las teclas generales.

<↑,↓>	Para moverse hacia arriba y hacia abajo por la ventana de ayuda, y las ventanas de programa y memoria.
<Re. Pág>	Para desplazarnos una página hacia atrás, o hacia adelante.
<Av. Pág>	En la ventana de memoria, el cursor se desplazará a la primera y a la última dirección de memoria, como veremos en el apartado de la Ventana de Memoria.
<ESC>	Para salir de la ayuda, cancelar cualquier edición o introducción de datos por teclado, o quitar un mensaje de error.
<F1>	Muestra la ventana de ayuda general.
<CTRL-F1>	Muestra la ayuda específica de cada ventana.
<F2>	Volcado de memoria a disco. Grabaremos en un fichero el programa contenido en memoria con extensión '.BIN.
<F3>	Volcado de disco a memoria. Cargaremos en memoria un programa contenido en un fichero con extensión '.BIN.
<ALT-F1>	Ayuda específica, para cada una de las ventanas.
<ALT-M>	Activa la Ventana de Memoria.
<ALT-R>	Activa la Ventana de Registros.
<ALT-P>	Activa la Ventana de Programa.
<TAB>	Como alternativa a las tres teclas de activación de cada ventana, podemos utilizar el Tabulador para saltar de ventana a ventana.
<ALT-V>	Copia un bloque de memoria, especificado por las direcciones de comienzo y fin del bloque, en otra posición de memoria.
<ALT-F>	Rellena un bloque de memoria, especificado por las direcciones de comienzo y fin del bloque, con el valor indicado.
<ALT-C>	Intercambia el contenido de dos bloques de memoria.

<CTRL-F10>	Provoca un reset de la máquina. Como consecuencia, el valor del PC se inicializará a 0, al igual que el flag Z, quedando intacto el contenido de la memoria.
<ALT-X>	Salir del Programa.

VENTANA DE MEMORIA

En la Ventana de Memoria se muestra el contenido de toda la memoria de la **MS**. Cada una de las líneas de la Ventana de Memoria contiene doce posiciones de memoria, excepto la última, que contiene *ocho*.

En la parte izquierda de la línea (en color verde) se muestra el *rango de direcciones* correspondiente a las doce posiciones de memoria. Dichas direcciones están especificadas en hexadecimal y en decimal, entre paréntesis.

Los dígitos en amarillo de la derecha indican los *contenidos* en *hexadecimal* de dichas posiciones de memoria.

En la parte inferior derecha de la ventana (en color azul) aparece la posición actual en la que se encuentra el cursor.

Teclas a utilizar en la Ventana de Memoria:

< ← >	Desplaza el cursor una posición a la izquierda.
< → >	Desplaza el cursor una posición a la derecha.
< ↑ >	Desplaza el cursor una línea hacia arriba, es decir, 12 posiciones hacia atrás.
< ↓ >	Desplaza el cursor una línea hacia abajo, es decir, 12 posiciones hacia adelante.
<INICIO>	Sitúa el cursor en la primera posición de la línea.
<FIN>	Sitúa el cursor en la última posición de la línea.
<Re. Pág>	Desplaza el cursor a la primera posición de memoria (dirección 0).
<Av. Pág>	Desplaza el cursor a la última posición de memoria (dirección 127).
<ENTER>	Edita la posición sobre la que está el cursor, permitiendo cambiar su valor. Se debe introducir un número hexadecimal, de cuatro cifras como máximo.

VENTANA DE PROGRAMA

En esta ventana, se reflejan los nemónicos en la memoria. Los números en verde de la izquierda indican la *dirección de memoria* a la que corresponde dicho contenido, el cual está expresado en hexadecimal y en decimal entre paréntesis. La posición en la que está el cursor lleva un símbolo '>' delante.

El número en amarillo de cuatro dígitos que va a continuacion indica el *contenido de la posición de memoria* en cuestión. Dicho contenido vendrá expresado en hexadecimal.

A continuación, y en color morado, se muestra ese mismo *contenido en binario*, separando el código de instrucción (dos dígitos) y sus dos operandos (de siete dígitos cada uno).

Por último, en color azul, tenemos la instrucción desensamblada, con los operandos especificados en hexadecimal (dos dígitos) y en decimal (tres dígitos).

Teclas a utilizar en la Ventana de Programa:

<↑>	Pasar a la instrucción anterior.
<↓>	Pasar a la siguiente instrucción.
<Re. Pág>	Retrocedemos una página, equivalente a 10 instrucciones. El número de instrucciones equivalentes a cada avance/retroceso de página dependerá del modo de vídeo en el que se esté ejecutando el programa emulador.
<Av. Pág>	Avanzamos una página, equivalente a 10 instrucciones. Al igual que en el caso anterior, dicha equivalencia dependerá del modo de vídeo seleccionado.
<INICIO>	Lleva el cursor a la dirección 0.
<FIN>	Lleva el cursor a la dirección apuntada por el registro PC (Contador de Programa):
<T>	Tracear. Ejecuta la instrucción apuntada por el registro PC (Contador de Programa). Esta opción nos permite ejecutar el programa instrucción a instrucción. Paso a paso.
<A>	Ensamblar. Esta opción sirve para poder introducir un programa en nemónicos. Para ello, nos colocamos en la posición deseada y pulsamos la tecla <A>. A continuación, se mostrará una línea verde, con un cursor morado, en el cual, podremos introducir la instrucción.
<G>	Ejecuta un programa desde la instrucción apuntada por el PC hasta pulsar <ESC> o hasta que encuentra un Break Point, si lo hay. Se aconseja poner un Break Point después de la última instrucción. De esta manera evitamos que se ejecuten instrucciones que estén a continuación de nuestro programa y que no pertenecen al mismo. No se actualizarán los contenidos de las ventanas (como consecuencia de la ejecución del programa) hasta que éste termine.

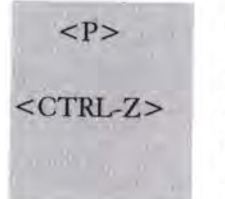
<P>	Al igual que la opción G, ejecuta un programa desde la instrucción apuntada por el PC hasta pulsar <ESC> o hasta que encuentra un Break Point, si lo hay. La diferencia entre ambas teclas radica en que con la opción P se van actualizando las ventanas (sus contenidos) según se está ejecutando el programa, cosa que no ocurre con G.
<CTRL-I>	Introduce por el teclado un nuevo valor para el PC, bien en decimal (tres dígitos) o bien en hexadecimal (dos dígitos), para lo cual introduciremos la letra 'H delante de la dirección.
<CTRL-F>	Establece un Break Point (Punto de Ruptura) en la instrucción actual, cambiando la línea al color rojo. Utilizaremos los Break Points como puntos de ruptura o parada en la ejecución de un programa. Podremos introducir tantos como deseemos.
<CTRL-B>	Desactiva el Break Point de la instrucción actual.
<ENTER>	Activa/Desactiva el Break Point (línea de color rojo) en la instrucción actual.
<Barra Espaciadora>	Establece la dirección de la instrucción actual como la dirección de inicio de ejecución. Es decir, actualiza directamente el valor del PC con la dirección actual.

VENTANA DE REGISTROS

En esta ventana se muestran los contenidos del *registro PC* (a la izquierda) y del *Flag Z* (a la derecha).

El contenido del PC está representado en hexadecimal y en decimal (en color rojo) entre paréntesis.

Teclas a utilizar en la Ventana de Registros:



<P>	Modifica el valor del registro PC (Contador de Programa) que apunta a la dirección de ejecución del programa.
<CTRL-Z>	Activa/Desactiva el Flag Zero. Dicho flag se modificará automáticamente cada vez que ejecutemos las instrucciones CMP, ADD y MOV y al hacer un reset.

12.13.8. Ejemplos de programas para la MS

PROGRAMA 1 (Resuelto)

Confeccionar un programa para calcular el factorial de números mayores que cero:

factorial n! = 1x2x3x4... xn

SOLUCIÓN

Se comienza con el factorial de índice 1 obteniendo el resultado (rdo) que será 1. Se continúa multiplicando el factorial ya calculado por el número actual (índice). Si el índice alcanza al número del que se desea obtener el factorial finaliza el programa.

factorial (índice) = rdo.

factorial (1) = 1

factorial (2) = 1 x 2 = 2

factorial (3) = 2 x 3 = 6

Se va multiplicando el factorial ya calculado y residente en rdo. por el número actual (índice). Se usa la variable auxiliar **cont**, puesto que no se puede emplear rdo. como operando y resultado a la vez.

rdo = rdo x índice (operación que deseamos realizar)

cont = rdo

rdo = cont x índice

índice = índice + 1

Listado del Programa

```
00              mov     uno, índice
01              mov     uno, rdo
02   bucle:     cmp     índice, a
03              beq     fuera          ;se sale del bucle y finaliza
04              mov     cero, cont
05              mov     rdo, num       ;multiplicación rdoxíndice
06   multi:     cmp     cont, índice
07              beq     fmul
08              add     num, rdo
09              add     uno, cont
0A              cmp     uno, uno
0B              beq     multi
0C   fmul:      add     uno, índice
0D              cmp     uno, uno
0E              beq     bucle          :salto incondicional
0F   fuera:     beq     fuera          ;fin
.........................................................
10   a:         word    0003
11   rdo:       word    0000
12   índice:    word    0000
13   cont:      word    0000
14   num:       word    0000
15   uno:       word    0001
16   cero:      word    0000
```

PROGRAMA 2

Realizar un programa con las instrucciones de la **MS** que calcule una potencia de números enteros, dando la base y el exponente.

PROGRAMA 3

Confeccionar un programa que cuente el número de posiciones de la memoria que contiene el dato 3B H, en el rango comprendido desde la dirección 20 H a la 40 H.

PROGRAMA 4

Desarrollar un programa que compare dos números situados en sendas posiciones de memoria e indique si el primero es mayor, menor o igual al segundo.

PROGRAMA 5

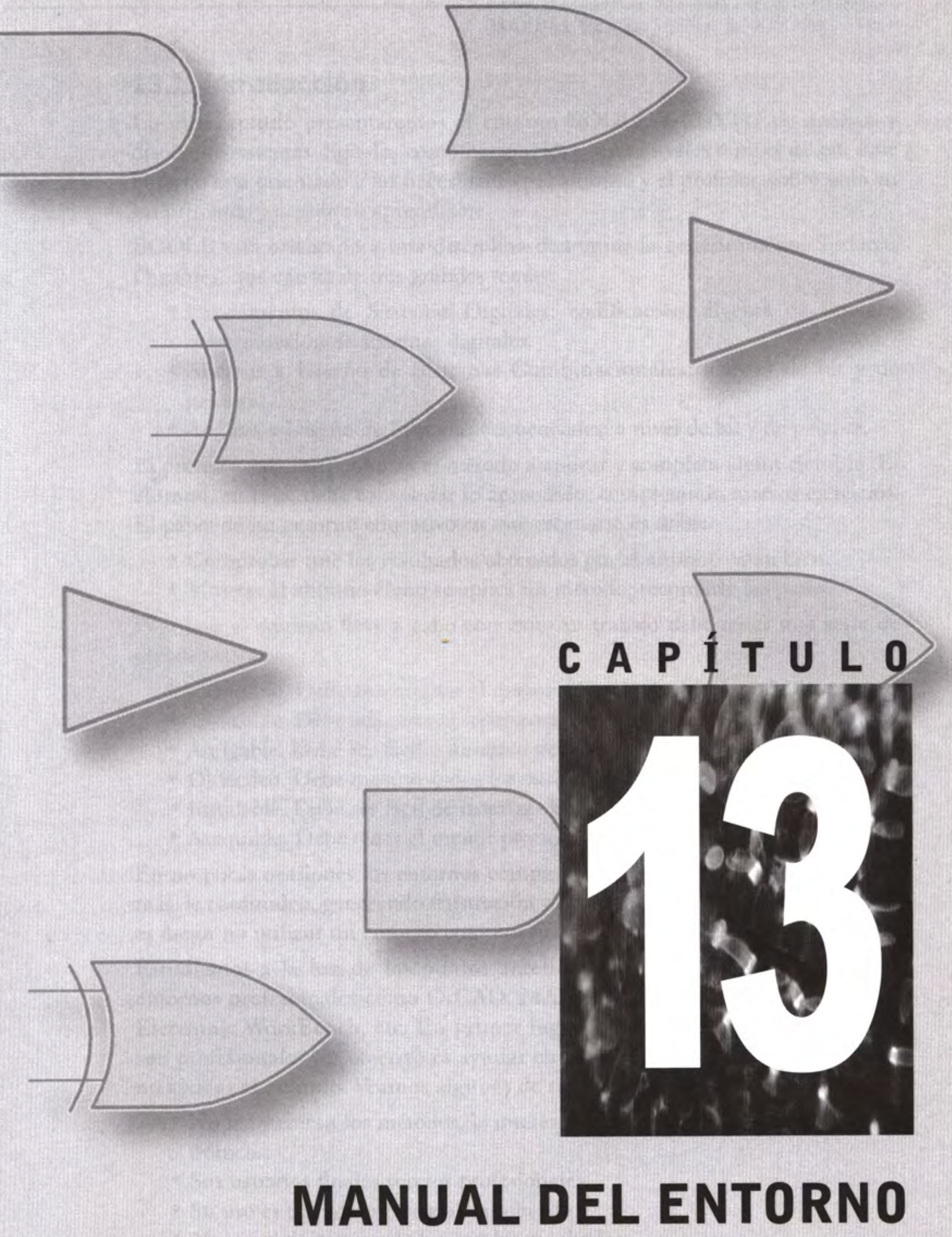
Diseñar un programa que rellene las 50 últimas posiciones de Memoria con el valor FF H.

PROGRAMA 6

Confeccionar un programa que calcule el perímetro de un rectángulo, proporcionando el valor de sus lados.

PROGRAMA 7

Desarrollar un programa que calcule el área de un cuadrado, conociendo el valor de su lado.



CAPÍTULO

13

MANUAL DEL ENTORNO BOOLE-DEUSTO

13.1. Introducción

En este capítulo presentaremos el entorno BOOLE-DEUSTO de análisis y diseño de sistemas digitales combinacionales y secuenciales a nivel de bit. Este entorno está orientado a las necesidades del alumno y el profesor, sobre todo en los primeros pasos de su aprendizaje.

BOOLE está orientado a una disciplina denominada genéricamente Sistemas Digitales, que consta de tres grandes temas:

- Fundamentos de Sistemas Digitales: codificación, álgebra de Boole y representación de sistemas digitales.
- Análisis y Diseño de Sistemas Combinacionales: a nivel de bit y de palabra.
- Análisis y Diseño de Sistemas Secuenciales: a nivel de bit y de palabra.

El profesor, en clase, enuncia el método a aplicar y completa algún ejemplo. El alumno, en casa, debe consolidar lo aprendido, completando nuevos ejercicios. El papel de un entorno educativo en este escenario es doble:

- Comprobar que los resultados obtenidos por el alumno están bien.
- Mostrar al alumno cómo se aplica un método; recordarle los pasos.

Para que el entorno lleve a cabo con éxito su trabajo debe tener una serie de características:

- Completo. Debe contemplar el mayor número posible de métodos.
- Utilizable. Debe adaptarse a los conocimientos y posibilidades del alumno.
- Amigable. Debe ser fácil e intuitivo de usar y visual en los resultados.
- Didáctico. Debe mostrar todos los pasos que le llevan a un resultado.
- Instalable. Debe ser fácil de instalar.
- Asequible. Debe tener el menor precio posible.

En no pocas ocasiones los entornos computacionales no ayudan al alumno; es más, le confunden, generando frustración y desconocimiento. A nuestro parecer es mejor no utilizar un entorno computacional si éste es inadecuado.

Estudiemos a la luz de los puntos anteriores la idoneidad didáctica de los entornos profesionales como OrCAD, MAX PLUS II, XILINX Foundation, Electronic WorkBench, etc. En primer lugar, recordemos que estos entornos son profesionales; su objetivo es ayudar en el diseño de circuitos electrónicos, no ayudar al alumno. Veamos algunas de sus características:

- No le interesan los métodos, le interesan sus resultados: los circuitos electrónicos.
- Sus usuarios finales son los profesionales.
- Su uso es tan potente como complicado.
- No son didácticos; sólo buscan los resultados.
- Son difíciles de instalar.
- Son generalmente caros.

Resulta tan evidente como sencillo que los objetivos de los entornos profesionales están alejados de las necesidades del profesor y del alumno en el aula. La tabla 13-1 compara ambos entornos, resaltando las diferencias.

Tabla 13-1

Comparación entre BOOLE y los entornos profesionales.

ENTORNO BOOLE-DEUSTO	ENTORNOS PROFESIONALES
Didáctico	Profesional
Necesidades del alumno	Necesidades del profesional
Hasta nivel de bit	Hasta nivel de sistema
Proyectos sencillos	Proyectos complejos
El alumno controla la herramienta	El usuario es dirigido por la herramienta
Interesa el proceso	Interesa el resultado
Sin instalación	Instalación compleja
Es muy fácil de usar	Críptico y difícil de usar
Es gratis y de libre distribución	Coste generalmente elevado
No tiene simulación temporal	Sí tiene simulación temporal
No permite la captura gráfica de un circuito	Permite la captura gráfica de un circuito

Todo lo anterior viene a fortalecer la necesidad de un entorno completo y didáctico en la disciplina de sistemas digitales: BOOLE-DEUSTO.

13.2. Aspectos básicos de uso del BOOLE-DEUSTO

El entorno BOOLE-DEUSTO es muy fácil de utilizar, es más, éste es uno de los objetivos principales, incluso más importante que cubrir una mayor parte de la asignatura. El entorno ha de ser útil al alumno y al profesor con poco esfuerzo.

Se puede decir que 15 minutos son más que suficientes para enseñar al alumno a utilizar su parte combinacional, y otro tanto para su parte secuencial.

En los próximos apartados se presentará en detalle el entorno BOOLE-DEUSTO, pero es ahora momento de recalcar los aspectos básicos:

- Los sistemas han de ser secuenciales o combinacionales, pero siempre a nivel de bit.
- Los sistemas combinacionales tienen como núcleo la tabla de verdad, mientras que los secuenciales son autómatas.
- Los nombres de las variables pueden ser cambiados.
- Los autómatas pueden ser de Moore o de Mealy.
- Al describir un sistema hay que empezar dándole nombre e indicando el número de variables de entrada y de salida.

- Al salir de una pantalla de captura de datos hay que pulsar siempre `Evaluar` (y `Salir`) para que los datos sean actualizados.
- Las funciones se cargan y se visualizan de una en una, utilizando la barra de desplazamiento.
- Las tablas de verdad y diagramas de Veitch-Karnaugh no deben tener huecos (se puede utilizar la opción `Completar con`).
- Se pueden imprimir los resultados tanto para sistemas secuenciales como combinacionales.
- Los elementos gráficos (circuitos lógicos y diagramas de transición de estados) pueden ser copiados al portapapeles, y de él a cuaquier documento.
- Todo sistema, ya sea combinacional o secuencial, puede ser guardado y cargado como un fichero.
- Las pantallas que tengan alguna complicación disponen de `Ayuda`.

Para describir el entorno BOOLE nos apoyaremos en ejemplos.

13.3. Instalación y uso

Para instalar BOOLE bastará con copiar el ejecutable, o con copiar y descomprimir el fichero .zip. La instalación no puede ser más sencilla, de hecho fue un requisito a la hora de diseñar el entorno. De este modo, al alumno le basta con crear una carpeta (o no), copiar el programa, hacer doble clic sobre él y empezar a trabajar.

En cuanto al uso, éste es libre para cualquier usuario, quedando expresamente prohibida su distribución comercial de cualquier modo sin el consentimiento de los autores (BOOLE está en el Registro de la Propiedad Intelectual).

Se solicita a todos los usuarios que se registren en la dirección electrónica *zubia@eside.deusto.es*, y así poder enviarles las nuevas versiones del entorno.

13.4. Sistemas combinacionales con BOOLE

Antes de pasar a BOOLE, recordemos qué es diseñar. El diseño de un sistema combinacional pasa por varias fases:

- Leer y entender el enunciado.
- Determinar las variables de entrada y salida.
- Obtener la tabla de verdad.
- Obtener las formas normales de cada variable de salida.
- Obtener los diagramas de Veitch-Karnaugh de cada salida.
- Simplificar cada V-K, obteniendo la expresión simplificada.
- Opcionalmente, reescribir la expresión anterior desde las puertas NAND o NOR.
- Obtener el circuito lógico.
- Implementar el circuito digital con circuitos integrados.
- Probar el circuito implementado.

Los dos primeros pasos deben ser dados por el alumno/profesor, y los dos últimos quedan fuera del BOOLE; pertenecen al laboratorio, con programas como Electronic WorkBench, OrCAD, etc. Centrémonos mediante un ejemplo en los seis pasos restantes.

13.4.1. Ejemplo 1 de sistema combinacional

El ejercicio que se propone consiste en diseñar el circuito transcodificador de BCD puro a BCD Aiken.

Las variables de entrada son cuatro, las mismas que de salida. Así pues la tabla de verdad a completar tendrá 16 filas y cuatro salidas.

La figura 13-1 muestra la primera imagen que el alumno ve al activar BOOLE; en esta pantalla elegirá la opción `Sistemas Combinacionales`.

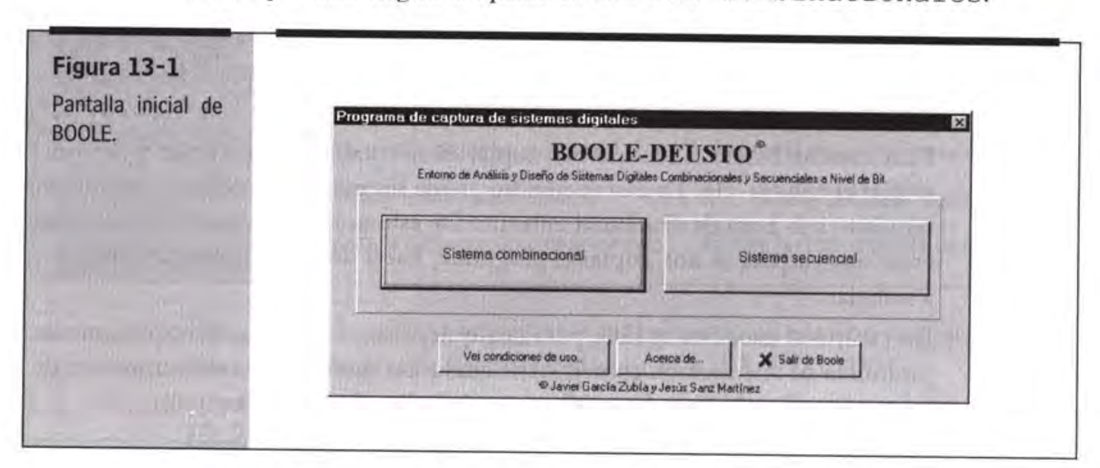


Figura 13-1
Pantalla inicial de BOOLE.

La siguiente imagen será la de la figura 13.2. En ella, el alumno obligatoriamente debe indicar el nombre del sistema, el número de variables de entrada y el número de variables de salida. Opcionalmente podrá dar un nombre a las variables, teniendo en cuenta que BOOLE asigna un nombre por defecto.

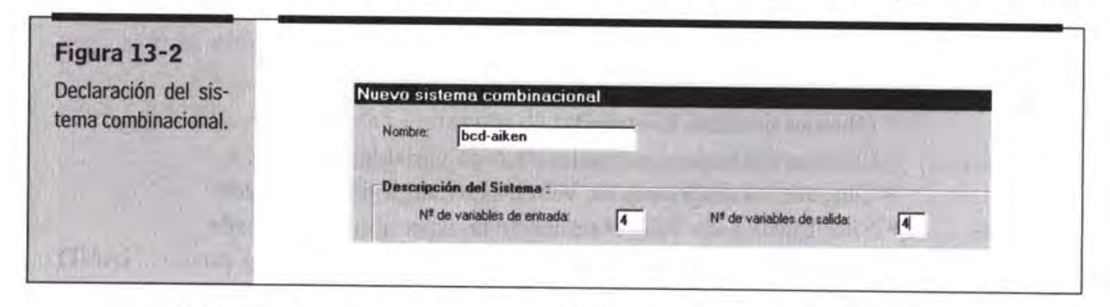


Figura 13-2
Declaración del sistema combinacional.

Una vez declarado el sistema, el alumno debe describirlo. La figura 13-3 muestra todas las opciones que ofrece BOOLE; en este caso optaremos por la opción `Tabla de Verdad Manual`.

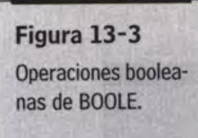
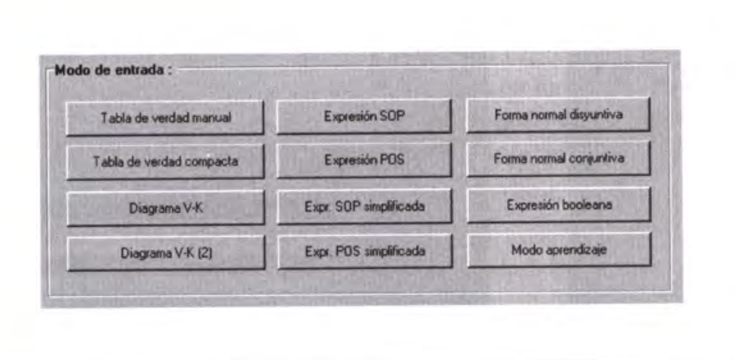


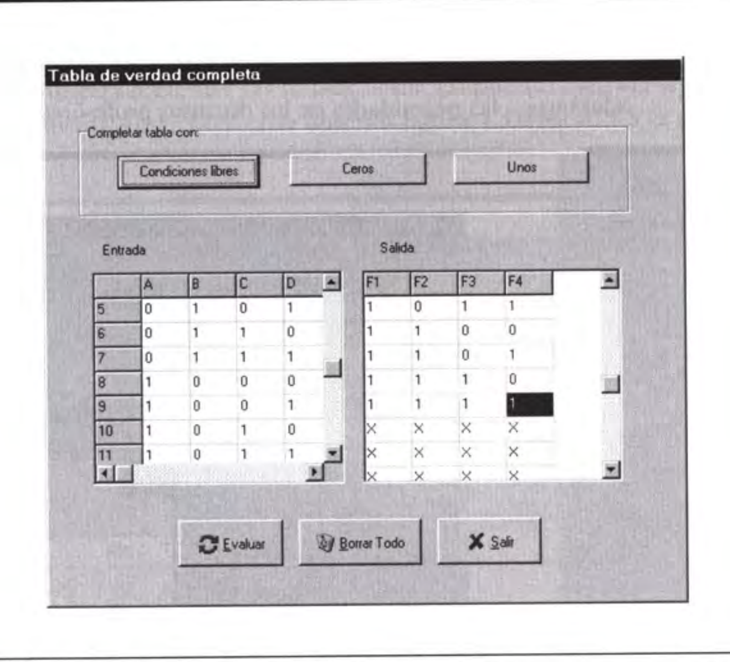
Figura 13-3
Operaciones booleanas de BOOLE.



Al activar esta opción el alumno verá las 16 filas ordenadas, debiendo escribir él las salidas mediante clics de ratón. Al ponerse encima de una casilla, cada clic de ratón cambia el valor booleano. Además, no hay que cumplimentar toda la tabla, se pueden escribir sólo los 1's y rellenar automáticamente el resto con 0's, o escribir sólo los 1's y 0's y rellenar el resto con condiciones libres.

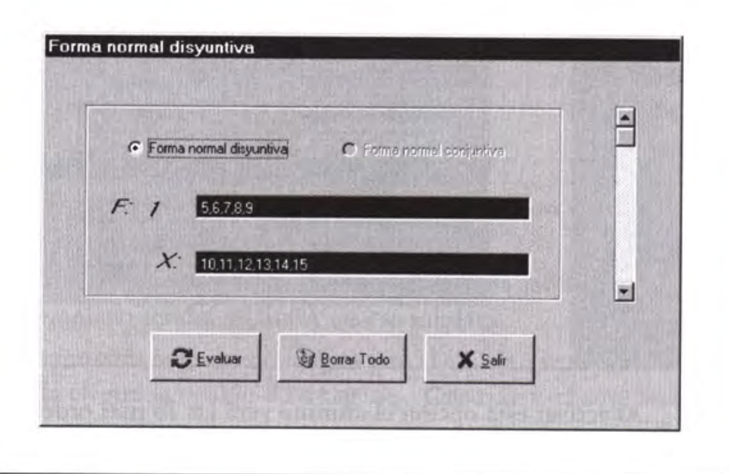
La figura 13.4 muestra la tabla de verdad del transcodificador. Para que el sistema sea cargado es necesario activar `Evaluar`, y luego `Salir`. A partir de este momento, todas las operaciones de BOOLE serán aplicadas a este sistema.

Figura 13-4
Tabla de Verdad Manual.



Al volver a la figura 13-3, si el alumno activara `Forma Normal Disyuntiva` obtendría la figura 13-5. En ella sólo se ve una salida, la primera; para ver las restantes basta con pulsar en la barra de desplazamiento de la derecha.

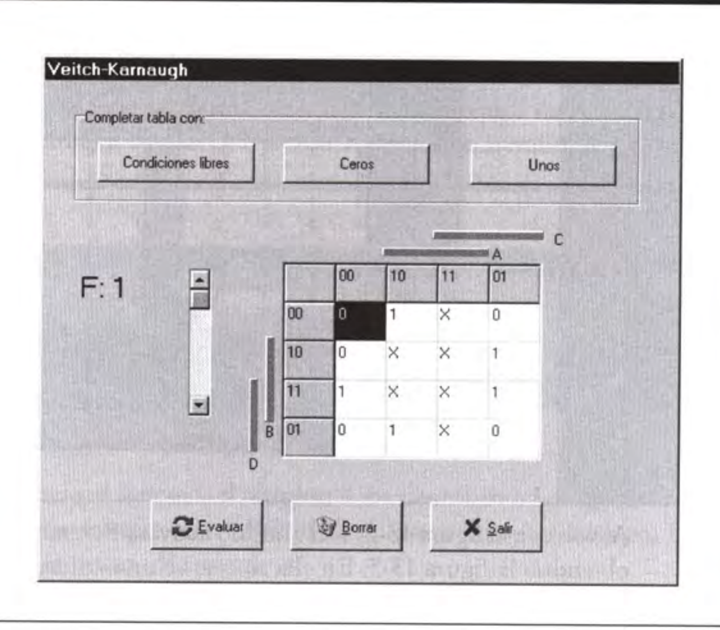
Figura 13-5
Forma Normal Disyuntiva.



Si el alumno quisiera, podría cambiar el sistema borrando los minitérminos de F1 y escribiendo otros nuevos. Pero sigamos con el sistema tal y como lo hemos descrito al principio.

Si después de pulsar `Salir` en la anterior figura, el alumno pulsara `V-K` en la pantalla principal, se encontraría con la figura 13.6. Para ver los V-K de las restantes salidas, bastará con pulsar en la barra de desplazamiento. Además de la opción `V-K`, el alumno dispone de `V-K 2`; con esta opción obtendría un V-K ordenado de distinta manera (según el código Gray). De esta forma, BOOLE busca adaptarse a las necesidades de los distintos profesores y alumnos.

Figura 13-6
Diagrama de Veitch-Karnaugh.



El anterior V-K tiene por objeto facilitar la simplificación visual de la función. Al pulsar `Salir` en la figura 13-6 y `Expr. SOP Simplificada` en la pantalla principal de la figura 13.3, el alumno obtendrá las expresiones simplificadas de cada una de las salidas. Dichas expresiones lo serán en forma de suma de productos (SOP), aunque también podría haber optado por POS. Las expresiones obtenidas son mínimas necesariamente (el algoritmo implementado es recursivo y exacto), aunque el conjunto no tiene por qué serlo, es decir, simplifica cada función por separado.

La figura 13-7 muestra la expresión mínima de la salida 2 (para ver el resto pulsaremos en la barra de desplazamiento).

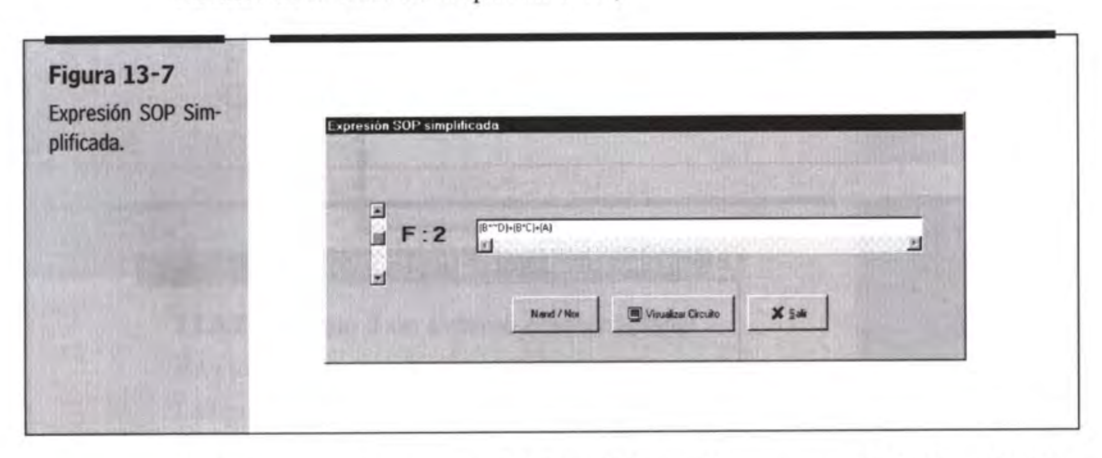


Figura 13-7
Expresión SOP Simplificada.

Si el alumno quisiera reescribir las salidas desde las puertas NAND o NOR, le bastaría con activar `Nand/Nor` en la anterior figura. La figura 13-8 es un ejemplo de ello.

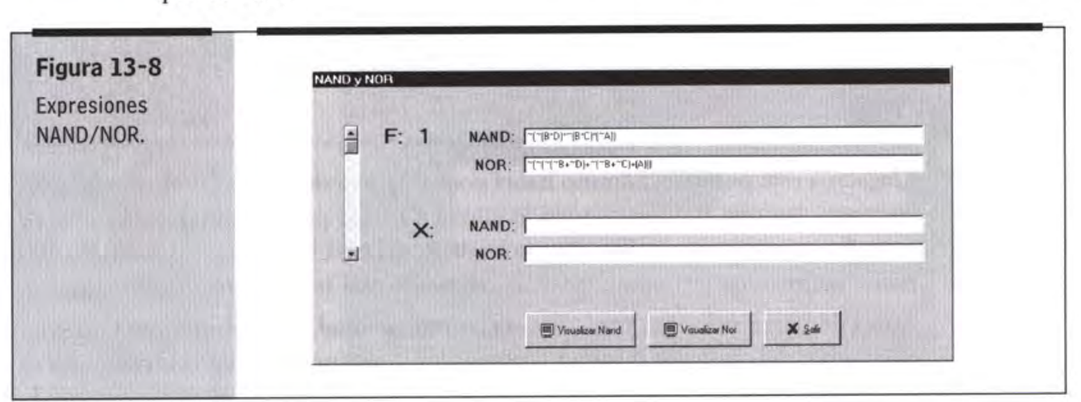


Figura 13-8
Expresiones NAND/NOR.

Para ver los circuitos lógicos el alumno deberá pulsar `Visualizar Circuito` en la figuras 13-7 o 13-8, resultando las figuras 13-9 y 13-10.

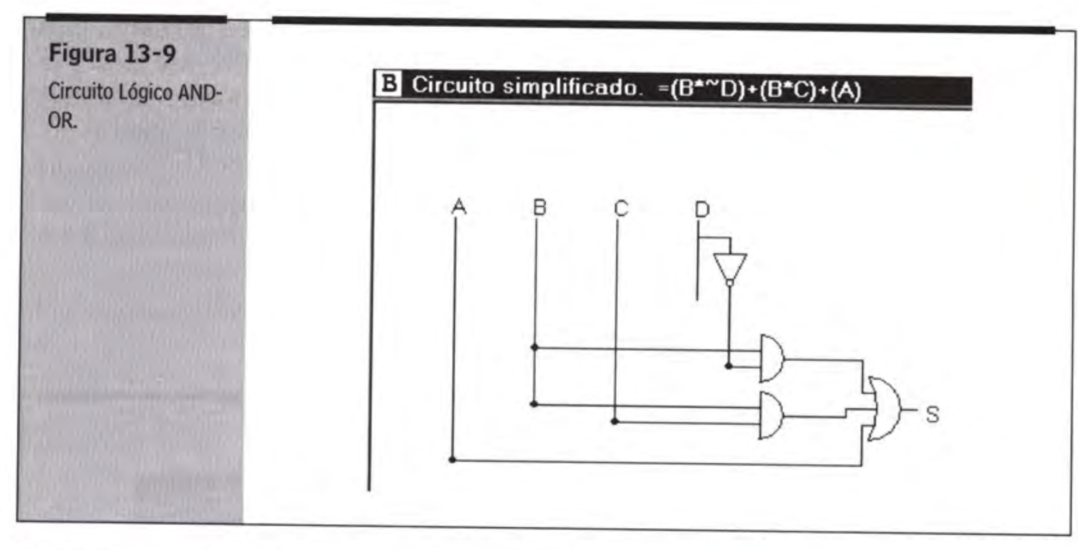


Figura 13-9
Circuito Lógico AND-OR.

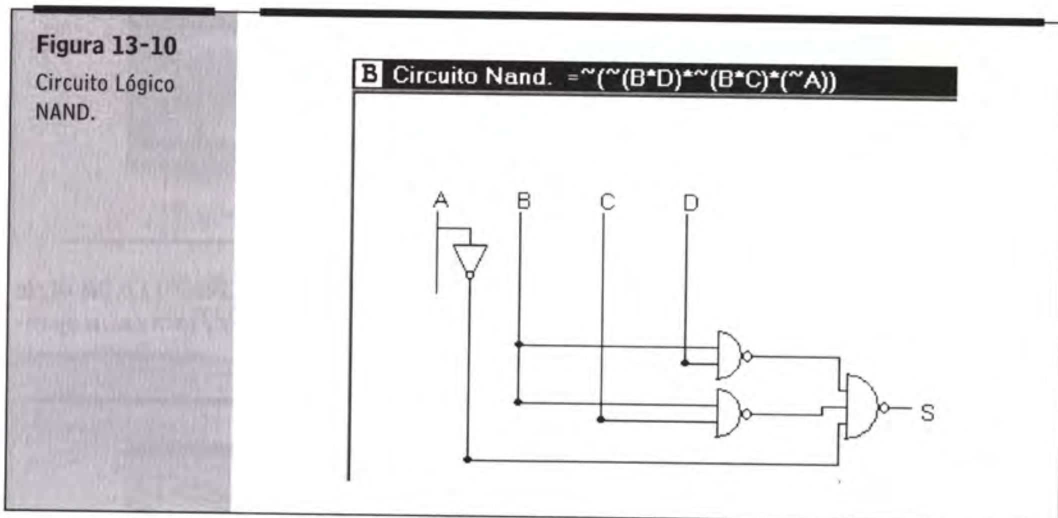


Figura 13-10
Circuito Lógico NAND.

Llegado a este punto, el alumno habrá completado todo el proceso de diseño paso por paso. Aunque si hubiera querido también habría podido introducir la tabla de verdad y seguidamente pulsar `Expr. SOP Simplificada`, ahorrándose los pasos intermedios; actitud que se debe evitar mientras se esté aprendiendo a diseñar.

Antes de pasar a diseñar otro sistema combinacional, el alumno podrá guardar el sistema, imprimir los resultados obtenidos u obtener el código correspondiente al sistema en OrCAD-PLD. La figura 13-11 muestra dichas opciones en la pantalla principal.



Figura 13-11
Servicios generales de BOOLE.



Si se optara por `Guardar PLD`, se obtendría un programa (ver figura 13-12) listo para ser compilado en OrCAD y finalmente grabado en la PAL elegida.

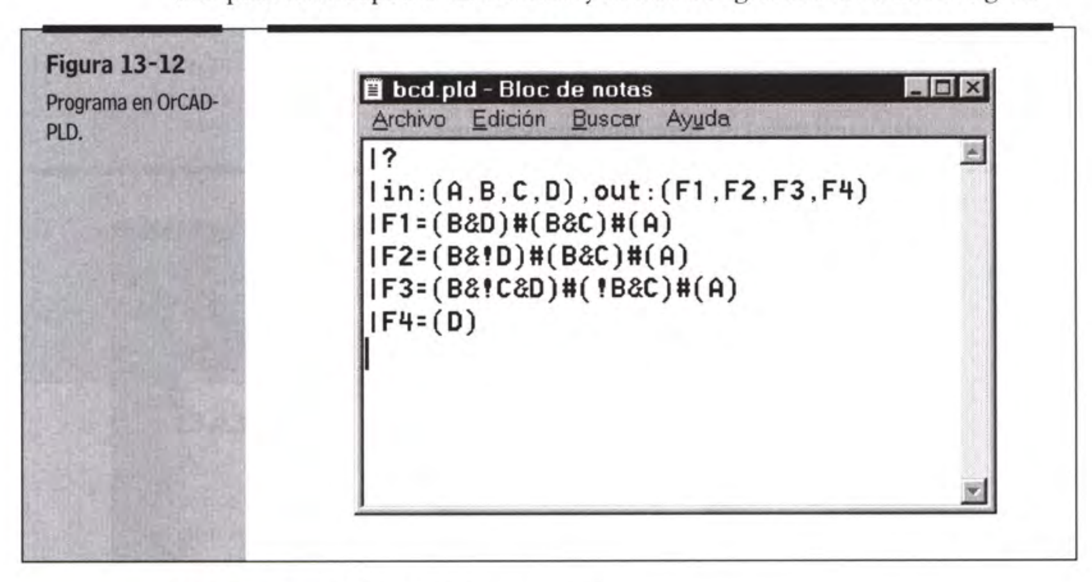


Figura 13-12
Programa en OrCAD-PLD.

13.4.2. Ejemplo 2 de sistema combinacional

En este caso se va a diseñar con BOOLE un codificador 8:3 con prioridad.

La figura 13-13 declara al sistema por su nombre y número de variables de entrada y salida.

Figura 13-13
Pantalla principal al completo.

Optaremos en este caso por describir el sistema con `Tabla de Verdad Compacta`. Como se puede ver en la figura 13-14, en este caso el alumno es responsable de la entrada y la salida. A cambio puede asociar condiciones libres en la entrada, es decir, escribir varias filas en una sola. De este modo, en vez de responder a 256 filas, el alumno escribirá solamente 8. Esta opción es muy cómoda (y peligrosa) en determinadas ocasiones.

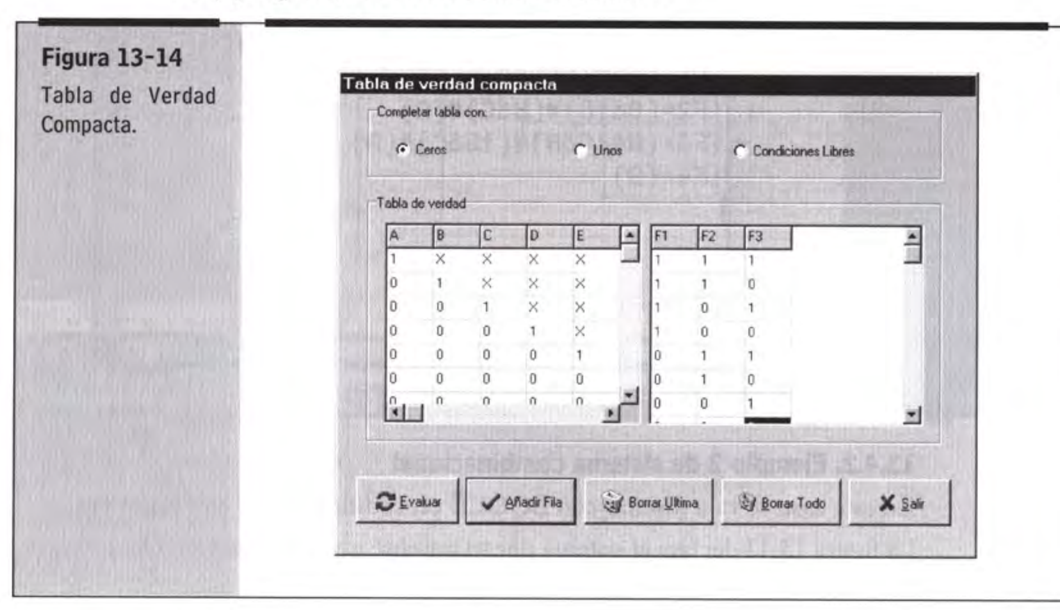


Figura 13-14
Tabla de Verdad Compacta.

Si al utilizar esta opción y las condiciones libres, el alumno no escribiera todas las filas, BOOLE las tomaría como condiciones libres, 0's ó 1's, según lo elegido en la parte superior de la figura 13-14.

Una vez cargado el sistema, el alumno puede pasar por todas las fases del ejemplo anterior o activar directamente `Expr. SOP Simplificada`. Las figuras 13-15 y 13-16 muestran la expresión simplificada de la salida 3 (la de menos peso) y el circuito lógico correspondiente, respectivamente.

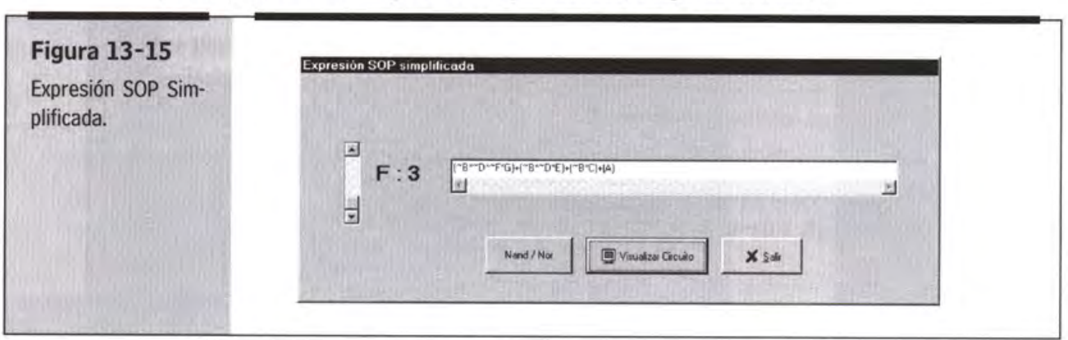


Figura 13-15
Expresión SOP Simplificada.

Los ejemplos han permitido describir los pasos a dar para diseñar un sistema combinacional. Ahora bien, la parte combinacional del entorno BOOLE permite otros usos de gran valor didáctico, como se ve a continuación.

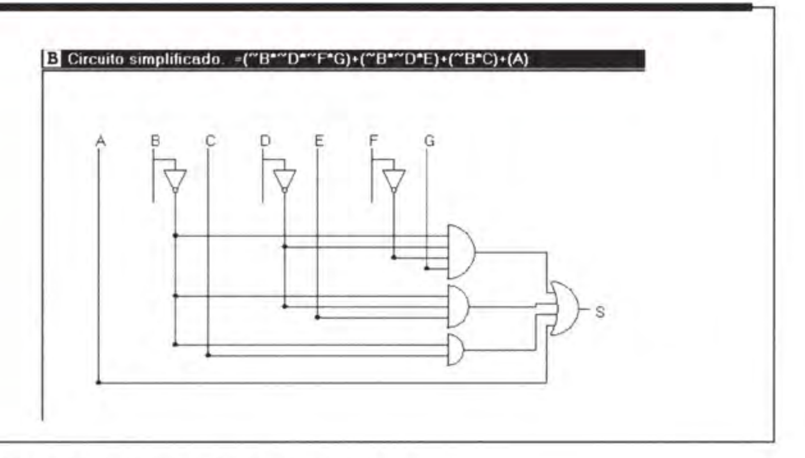


Figura 13-16
Circuito Lógico AND-OR.

13.4.3. Análisis de sistemas combinacionales

En el caso de sistemas combinacionales, y en general, se puede decir que analizar es seguir el camino inverso de diseñar, es decir, dado un circuito o unas ecuaciones obtener su tabla de verdad para comprobar su idoneidad. El primer caso queda fuera de los objetivos del BOOLE (cae dentro de los entornos profesionales: Electronic WorkBench, OrCAD, etc.), mientras que el segundo sí es contemplado por BOOLE.

Por ejemplo, el alumno puede comprobar si las expresiones por él obtenidas se comportan como un circuito de complemento a 9 (C3-0) para una entrada BCD (E3-0).

C3=(~E3*~E2*~E1) C1=(E1)

C2=(E2*~E1)+(~E2*E1) C0=(~E0)

Para comprobar dichas ecuaciones, el alumno habrá declarado el sistema con su nombre, 4 entradas y otras tantas salidas, y seguidamente activará `Expresión Booleana`. Para introducir las expresiones booleanas habrá que seguir las indicaciones que aparecen en la `Ayuda` de la pantalla. Introducidas una a una las cuatro funciones (utilizando la barra de desplazamiento) se obtiene la figura 13-17.



Figura 13-17
Captura de Expresión Booleana.

Después de pulsar `Evaluar` y `Salir` el alumno volverá a la pantalla principal, donde al activar `Tabla de Verdad Manual` obtendrá la tabla correspondiente a dichas ecuaciones, comprobando su validez. La figura 13-18 muestra la tabla de verdad de las ecuaciones anteriores y su validez.

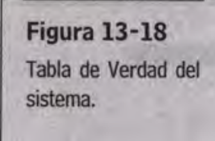
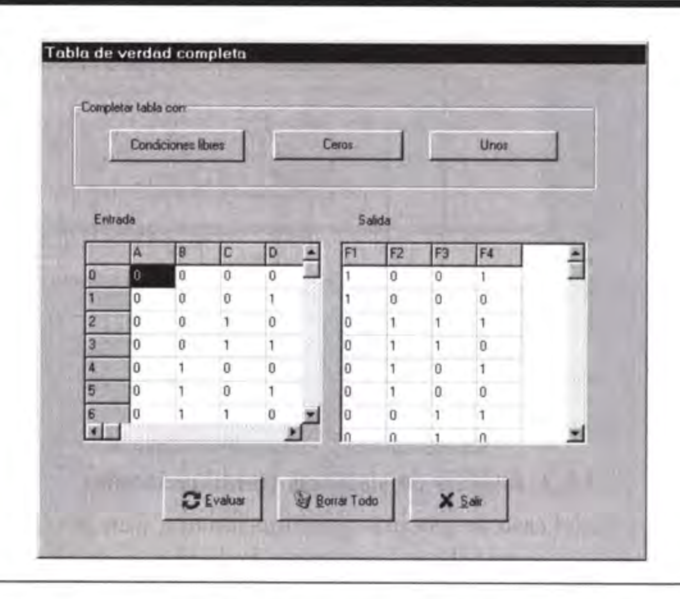


Figura 13-18
Tabla de Verdad del sistema.



BOOLE permite otros caminos de análisis, bastará con que el alumno introduzca el sistema mediante una técnica de representación y active otra para observar el resultado. De hecho, y visto así, análisis y diseño se distinguen en el punto de vista, y no en los métodos o técnicas de representación utilizados.

13.4.4. Simplificación de funciones

En el proceso de aprendizaje del alumno un paso costoso para él es aprender a simplificar funciones booleanas. El método en sí no es difícil de explicar, aunque sí lo es su aplicación. Como método heurístico que es, sólo se aprende practicando: dada una función, el alumno la simplifica y comprueba su acierto.

Visto desde el punto de vista de la clase, el profesor explica los pasos, completa algún ejercicio y deja el resto del trabajo al alumno. En este escenario BOOLE es de gran ayuda, tanto para el profesor como para el alumno.

Si el alumno eligiera en la pantalla de sistemas combinacionales la opción `Modo Aprendizaje` (una vez dado nombre y declarado el número de variables de entrada y salida), vería que aparecen el V-K del sistema cargado y una línea para que escriba la expresión que él crea mínima. BOOLE analiza lo introducido, indicando al alumno:

- la expresión es incorrecta (no se corresponde con el V-K),
- la expresión es correcta, pero no mínima (se corresponde con el V-K, pero no es todo lo mínima que se puede)
- la expresión es correcta y mínima (acierto).

Las figuras 13-19, 13-20 y 13-21 muestran un V-K con tres respuestas distintas. Para escribir las expresiones el alumno deberá seguir los consejos de la `Ayuda` y pulsar `Evaluar`.

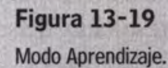


Figura 13-19
Modo Aprendizaje.

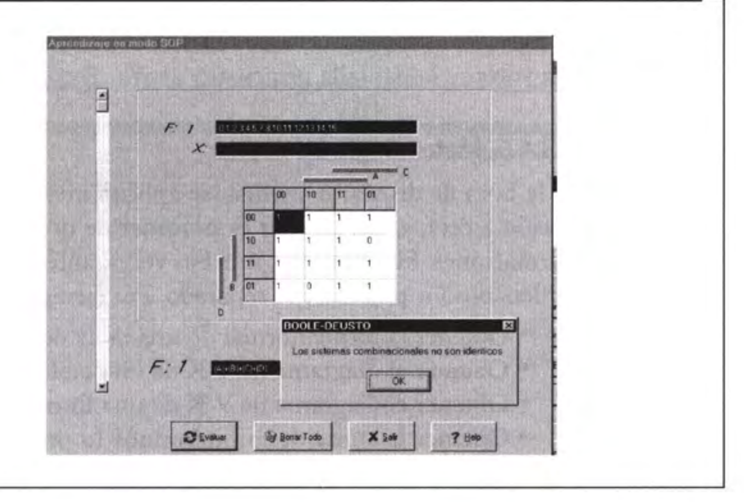


Figura 13-20
Modo Aprendizaje.

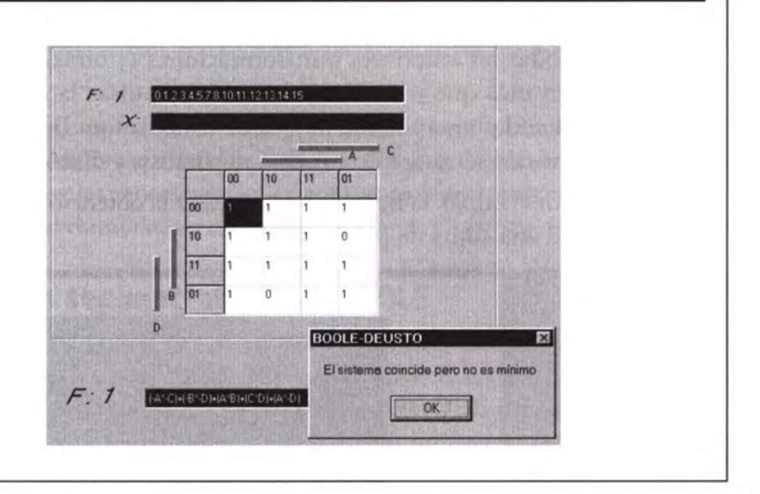
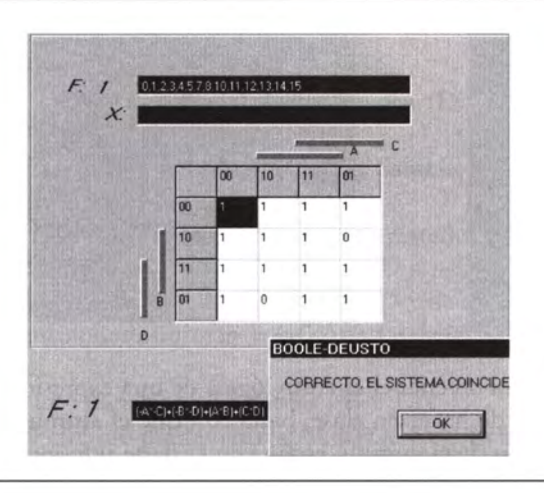


Figura 13-21
Modo Aprendizaje.



Como se puede ver en las figuras, BOOLE no muestra el resultado correcto, sólo informa al alumno de lo correcto de su opción. Si quisiera ver el resultado, debería volver a la pantalla principal y activar `Expr. SOP Simplificada`.

13.4.5. Metodología booleana

A la hora de diseñar o analizar se aplican métodos para pasar de una representación a otra, y, así, diseñar es proceder de una forma ordenada en estas transformaciones. Sin embargo, muchas veces, antes de que el alumno dé estos pasos ordenados los practica por separado. Por ejemplo:

- Obtener la forma normal de una tabla de verdad, y viceversa.
- Obtener el diagrama de V-K de una tabla de verdad, y viceversa.
- Obtener el diagrama de V-K de una forma normal, y viceversa.
- Obtener la suma de minitérminos (o producto de maxitérminos) de una tabla de verdad, y viceversa.
- Obtener la expresión NAND/NOR de una expresión SOP o POS.
- La obtención del circuito lógico de una expresión booleana.

Todas las anteriores transformaciones (y otras) son contempladas por BOOLE, sin más que introducir el sistema y pulsar la opción deseada. Es bajo este uso cuando toma sentido la idea de Calculadora Booleana asociada a BOOLE (además de ser una herramienta de análisis y diseño).

Por ejemplo, la figura 13-22 muestra la obtención de las expresiones NAND/NOR de una suma de productos.

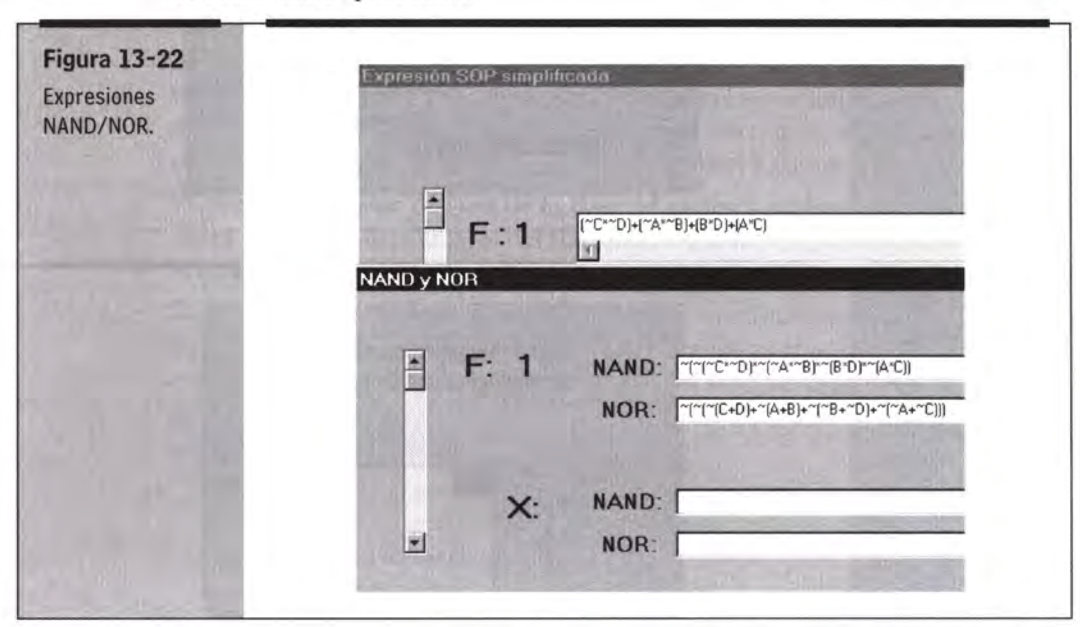


Figura 13-22
Expresiones NAND/NOR.

Para obtener el circuito lógico de una expresión, algo tan sencillo como incómodo de hacer en clase, basta con que el alumno llegue a la pantalla de visualización de circuitos y escriba en la parte inferior la ecuación a visualizar. Al activar

`Visualizar`, BOOLE le mostrará al alumno el circuito lógico de la expresión introducida. La figura 13-23 muestra el circuito lógico de una expresión cualquiera.

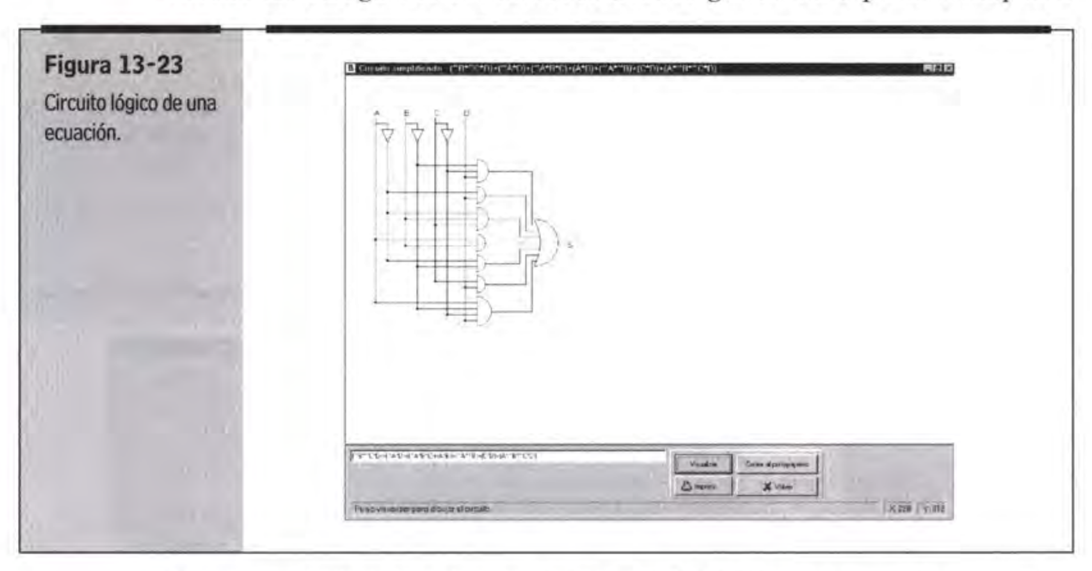


Figura 13-23 Circuito lógico de una ecuación.

Con estos sencillos ejemplos queda claro que BOOLE es un excelente compañero del alumno en sus primeros pasos metodológicos, cuando se acerca por primera vez al álgebra de Boole y las representaciones booleanas. Es probable que este interés del alumno sea poco duradero en el tiempo (se aprende rápido), pero será intenso.

13.5. Sistemas secuenciales con BOOLE

BOOLE permite analizar y diseñar sistemas secuenciales a nivel de bit, llamados también autómatas. El comportamiento de BOOLE con autómatas es distinto al visto para combinacionales. En este caso BOOLE no permite modificar el sistema en cualquiera de las fases, ya que esto no tiene sentido y no aporta nada didácticamente al entorno.

Las fases de diseño de un autómata son:

- Leer y entender el enunciado.
- Determinar las variables de entrada y de salida.
- Obtener el Diagrama de Transición de Estados (DTE), ya sea Moore o Mealy, indistintamente.
- Obtener la Tabla de Transición de Estados y Salidas.
- Obtener la Tabla de Codificación de Estados.
- Obtener la Tabla Codificada de Transición de Estados y Salidas.
- Obtener la Tabla de Excitación de Biestables D o J-K, indistintamente.
- Obtener los diagramas de V-K de cada J-K o D y de cada salida.
- Obtener las expresiones mínimas de las entradas de los biestables (J-K o D) y de las salidas.
- Obtener el circuito lógico con biestables J-K o D.
- Implementar el circuito digital correspondiente con circuitos integrados.
- Probar el circuito implementado.

Al igual que para sistemas combinacionales, los dos primeros pasos y los dos últimos no son contemplados por BOOLE, el resto sí, casi al completo.

13.5.1. Ejemplo 1 de sistema secuencial

El circuito a diseñar debe comportarse como un sumador serie. El sistema recibe en serie dos entradas A y B y ofrece también en serie en la salida la suma de dichas entradas.

Lo primero que el alumno debe hacer es elegir `Sistema Secuencial` (ver figura 13.24).

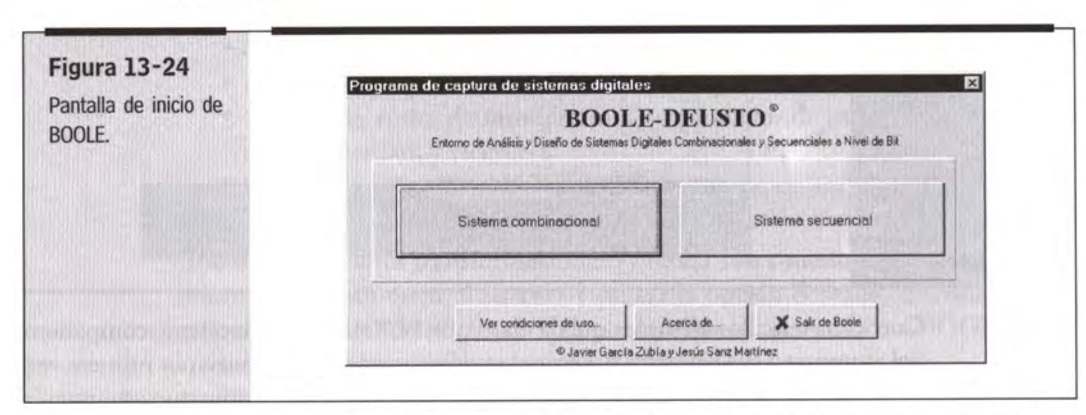


Figura 13-24
Pantalla de inicio de BOOLE.

La imagen que el alumno verá es la de la figura 13-25. Quizá el cuadradito con las flechas y el círculo quede fuera de la mesa de trabajo, pero el alumno puede seleccionarlo y arrastrarlo hasta ella.

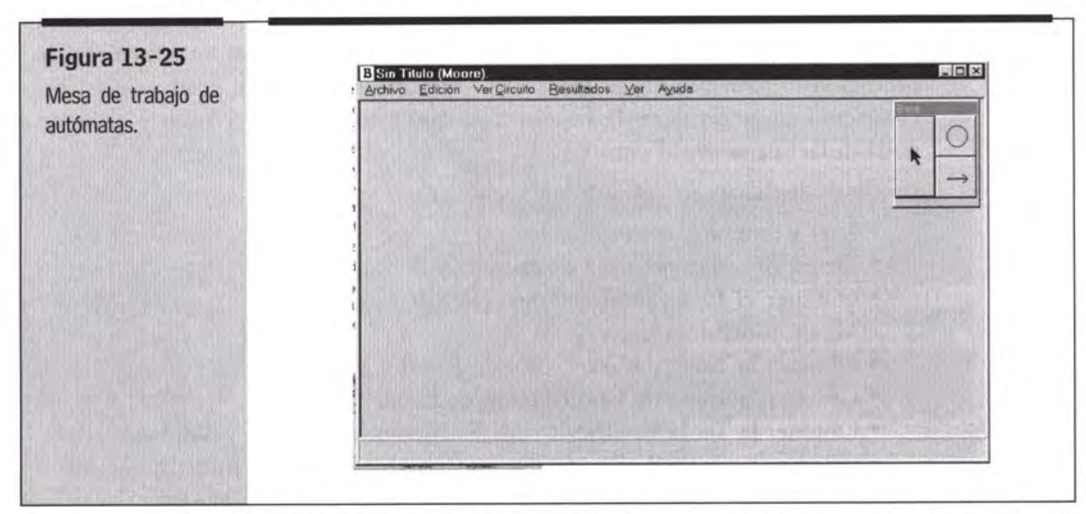


Figura 13-25
Mesa de trabajo de autómatas.

Seguidamente, el alumno debe declarar el autómata a diseñar. Para ello desplegará el menú `Archivo`, y en él elegirá la opción `Nuevo`, apareciendo la pantalla de la figura 13-26. El autómata a diseñar tiene dos entradas y una salida, y será descrito como Moore. Pulsando `Aceptar`, BOOLE vuelve a mostrar la mesa de trabajo.

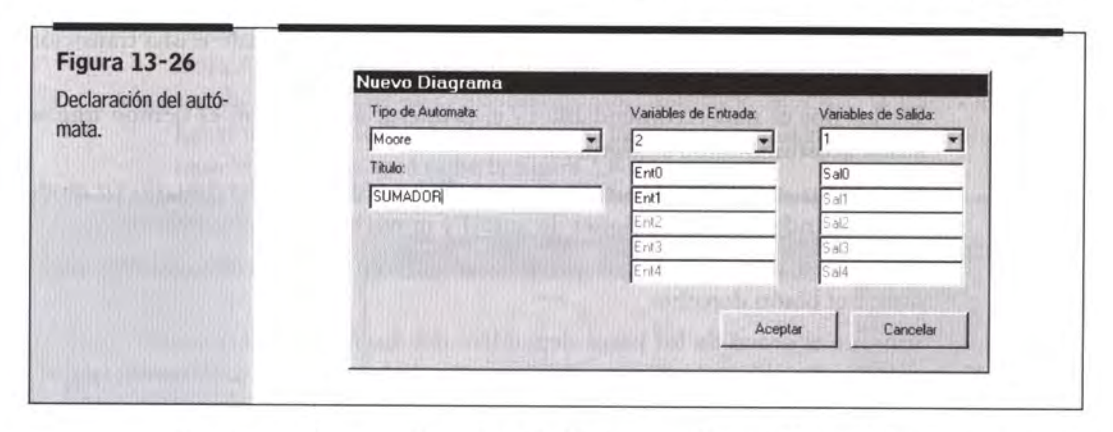


Figura 13-26

Declaración del autómata.

Una vez en la mesa de trabajo el alumno puede introducir el DTE gráficamente. En este aspecto BOOLE se muestra muy ágil y cómodo; de manera que cargar el DTE es tan fácil como vistoso.

Recordemos que un autómata tiene estados, transiciones, entradas y salidas, así BOOLE ofrece el círculo (estado) y la flecha pequeña (transiciones). La flecha grande sirve para seleccionar un estado o una transición. Veamos la captura del DTE del autómata de Moore de un sumador serie.

Por ejemplo, lo primero es situar los cuatro estados del autómata, seleccionando el círculo y haciendo simplemente clic con el ratón en la posición deseada. Así se obtiene la figura 13-27.

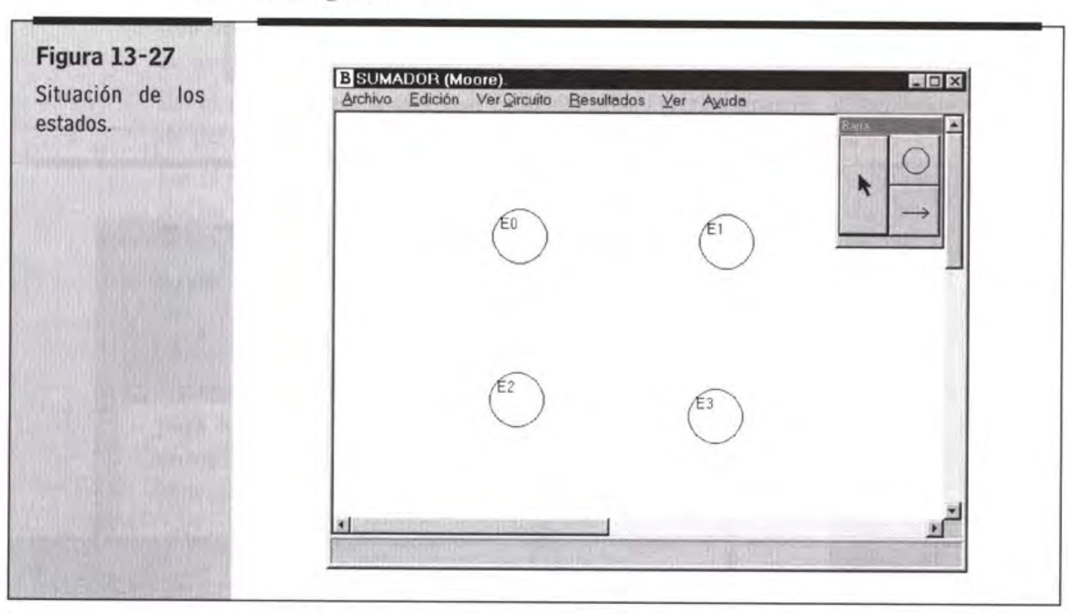


Figura 13-27

Situación de los estados.

Antes de seguir digamos cómo pasar de flecha grande a pequeña o círculo. Hay dos formas:

- Normal: basta con ir al cuadrado y seleccionar con el ratón.

- Rápido: al pulsar el botón derecho tras colocar un estado o una transición se pasa a la flecha grande.

De los dos el más recomendable es el primero, aunque con el tiempo uno se acaba acostumbrando al segundo.

Si al situar un estado quisiéramos moverlo bastaría con seleccionarlo en modo flecha grande (las dos opciones de antes) y moverlo a la nueva posición.

Para borrar un estado habrá que seleccionarlo en modo flecha grande y luego pulsar el botón derecho.

Aunque el orden de los pasos dependerá del diseñador y del ejercicio en particular, parece lógico seguir con la signación de las salidas a cada estado (por ser un autómata de Moore). Los estados son cuatro S0C0, S1C0, S0C1 y S1C1, el primero es cuando la suma es 0 y el acarreo de salida tambien, el resto de estados se entiende del mismo modo. Para asignar la salida al estado habrá que seleccionarlo en modo flecha grande y hacer doble clic sobre el estado. Al hacer esto aparece una pantalla en la que se escribirá el nombre del estado y las salidas asignadas (ver figura 13-28). La figura 13-29 muestra al autómata tras la asignación de las salidas a los estados.



Figura 13-28
Etiquetado de un estado Moore.

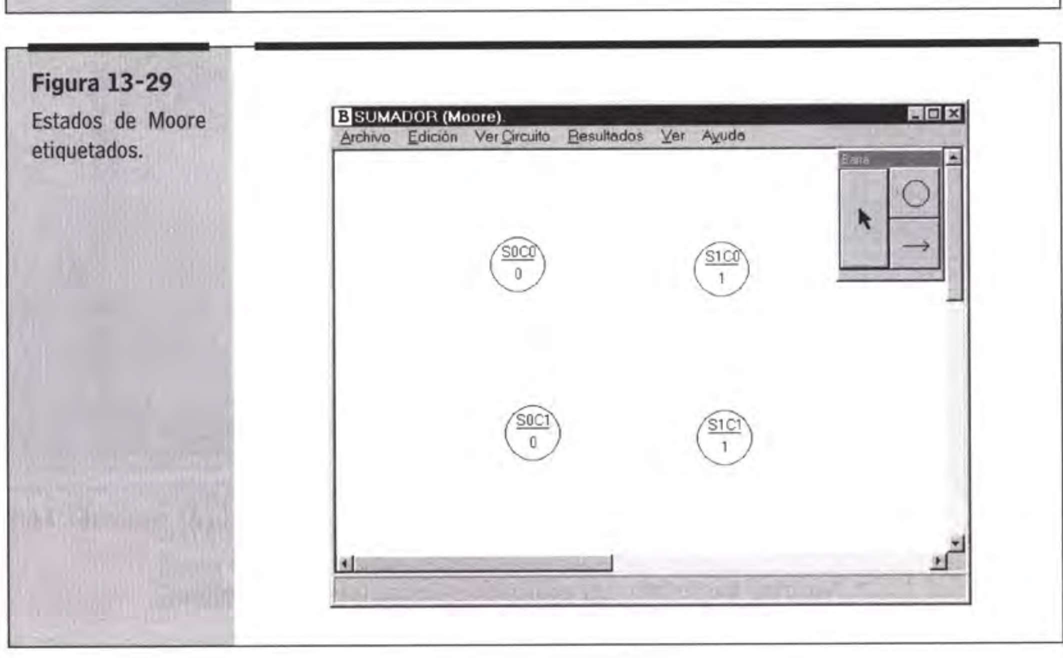


Figura 13-29
Estados de Moore etiquetados.

Situados los estados con su nombre y salida, el alumno debe dibujar las transiciones. Simplemente basta con seleccionar la flecha pequeña, hacer clic en el estado origen y clic en el estado destino; dibujando BOOLE la transición. Para hacer un autolazo basta con hacer los dos clics en el mismo estado. Acabado este paso el autómata será el de la figura 13-30.

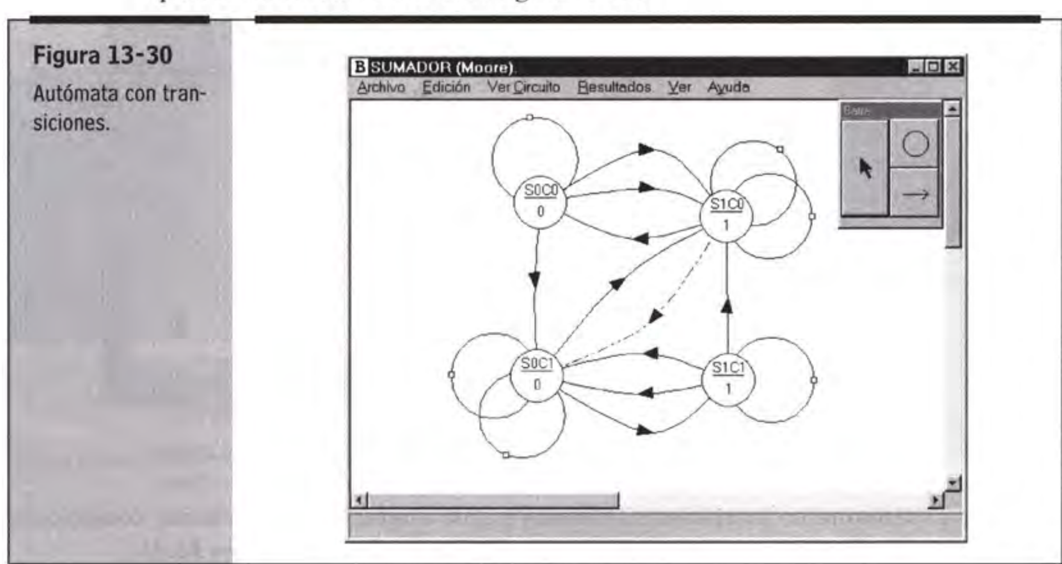


Figura 13-30
Autómata con transiciones.

Al situar las transiciones pueden darse dos situaciones incómodas: que una transición quede encima de otra o que no se vea la punta de flecha de la transición, en ambos casos la solución pasa por mover las transiciones.

En el primer caso, basta con estar en modo flecha grande, seleccionar la punta de flecha (la flecha del autolazo es un cuadrado) y arrastrarla.

En el segundo caso (¿dónde está la punta de flecha?) habrá que pasar a modo flecha grande, seleccionar el estado destino y moverlo, pues la punta de flecha estará debajo de él. Una vez que la flecha esté a la vista, se procede como en el párrafo anterior.

Todo lo dicho puede parecer complicado, pero es muy sencillo acostumbrarse al entorno de dibujo.

En este momento, sólo resta asociar las entradas a las transiciones. Al igual que para los estados basta con seleccionar la flecha, hacer doble clic y escribir la entrada de la transición (ver figura 13-31). Al acabar, el autómata tendrá el aspecto de la figura 13-32.

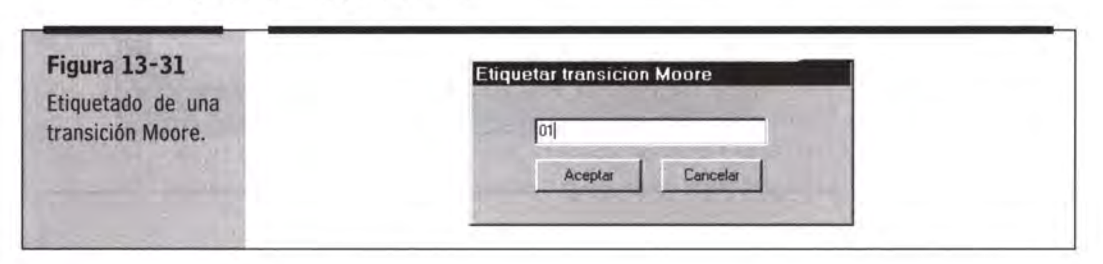


Figura 13-31
Etiquetado de una transición Moore.

Figura 13-32
Autómata de un sumador serie.

Aunque no es el caso, el alumno puede asociar a una transición condiciones libres en la entrada, como por ejemplo se muestra en la figura 13-33.

Figura 13-33
Uso de condiciones libres.

En este momento, el autómata de Moore ya está descrito mediante su DTE, y el alumno ya puede pasar a ver los pasos del diseño. Antes de diseñar se pueden activar dentro del menú `Resultados` las opciones `Verificar Corrección` y `Verificar Completitud`, que nos indican si hemos olvidado alguna transición, o si alguna de ellas está duplicada. La figura 13-34 nos muestra el resultado de dichas verificaciones.

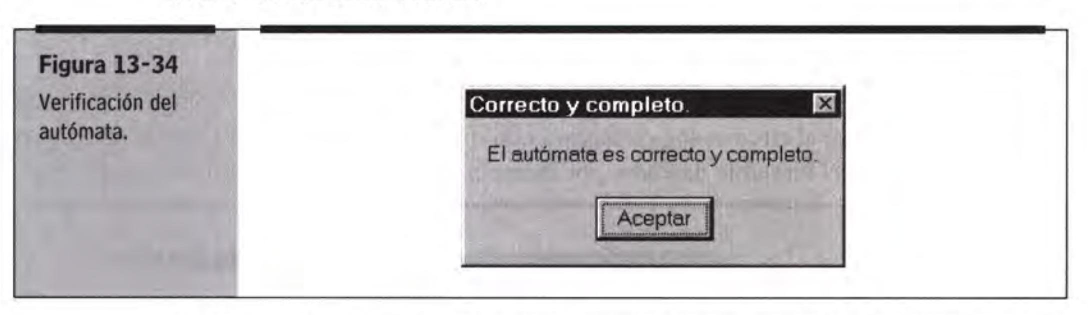


Figura 13-34
Verificación del autómata.

En el menú de `Resultados` podemos ver las opciones del diseño: `simular` el autómata, intentar `reducir` el autómata, obtener el autómata de `Mealy equivalente` y `diseñar` el autómata. Dejemos las tres primeras opciones para más adelante y activemos `Diseño`. En este caso, BOOLE ofrece todas las tablas del diseño y las expresiones mínimas obtenidas en una sola pantalla, la que aparece en la figura 13-35.

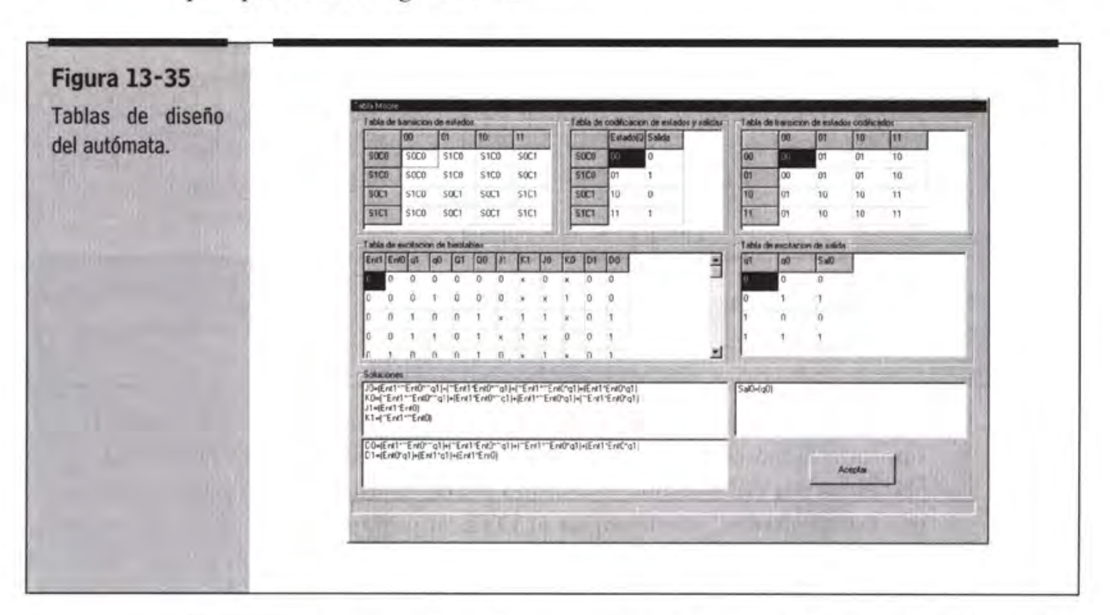
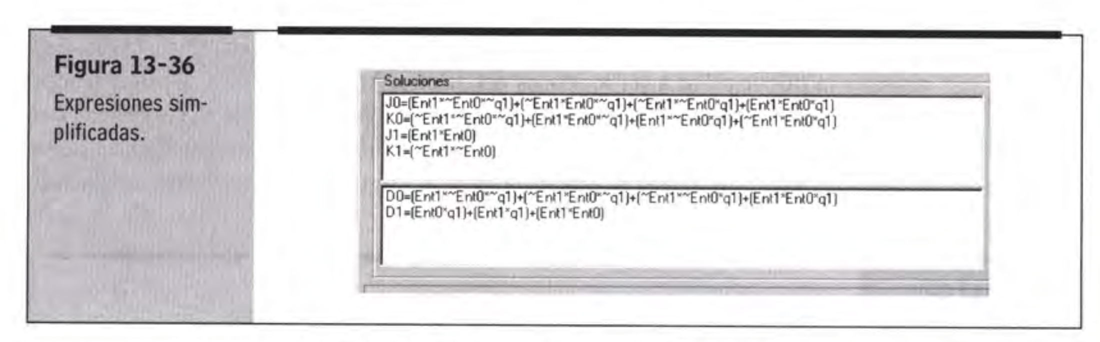


Figura 13-35
Tablas de diseño del autómata.

De la anterior imagen cabe aclarar que en la Tabla de Excitación de Biestables la q minúscula representa al estado actual, mientras que Q mayúscula representa al estado futuro, es decir, Q(t) y Q(t+1) o Q(t-1) y Q(t). La figura 13-36 muestra detallada la zona de las expresiones simplificadas.

Figura 13-36
Expresiones simplificadas.



Para ver el circuito lógico bastará con que el alumno vaya al menú `Circuito` y elija el biestable deseado, por ejemplo J-K, obteniéndose la figura 13-37.

Figura 13-37
Circuito lógico del autómata.

Siguiendo estos pasos el alumno habrá podido obtener no sólo el circuito lógico, sino que habrá observado los distintos pasos (y comprobado la bondad de los obtenidos por él). Además, el alumno puede guardar el autómata, puede imprimir los resultados, puede copiar el DTE al portapapeles (y de ahí a un documento, con lo que BOOLE vale para documentar ejercicios) y puede obtener el correspondiente programa en OrCAD-PLD.

13.5.2. Ejemplo 2 de sistema secuencial

Planteemos ahora el autómata de Mealy capaz de detectar las secuencias 1010 o 110 permitiendo solapamientos y cambios entre secuencias. Dispone de una entrada serie y de una salida que indica si se ha detectado una de las dos secuencias.

El DTE correspondiente al anterior enunciado es el de la figura 13-38.

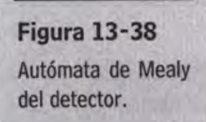
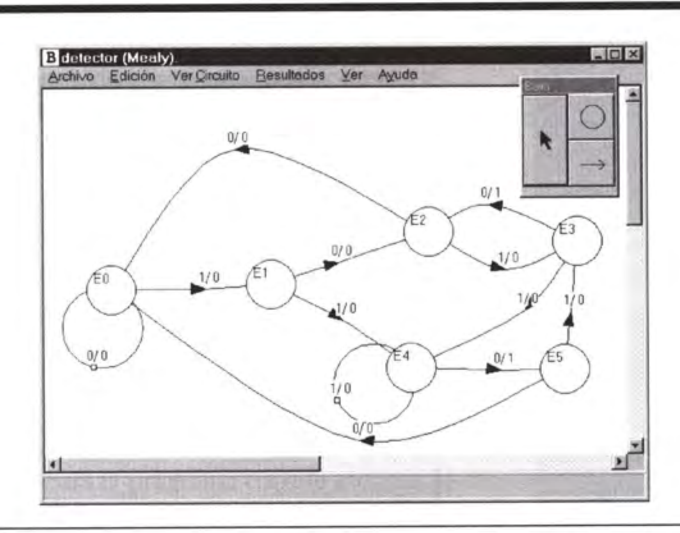
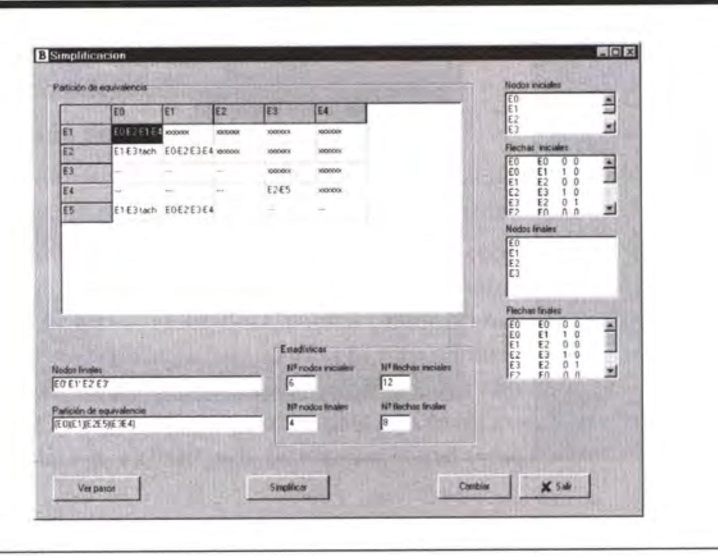


Figura 13-38
Autómata de Mealy del detector.



El alumno puede intentar reducir el anterior autómata de 6 estados. Basta con seleccionar `Reducción` en el menú `Resultados`. BOOLE le muestra al alumno una pantalla en la que elegirá si quiere ver los pasos de la reducción (`Ver Pasos`) o sólo ver los resultados (`Simplificar`). Si el alumno optara por la primera opción, deberá guiar el proceso con sucesivos clics de ratón sobre `Continuar`, y así ir viendo cómo se completa la tabla de reducción. Tras varios clics BOOLE ofrece el resultado de la posible reducción, que en este caso deja al autómata en cuatro estados. Si en la figura 13-39, tras la reducción, se optara por `Cambiar`, el propio BOOLE sustituirá el DTE antiguo por el nuevo, quedando éste a la vista. Si hiciera falta el alumno podría reordenar o modificar lo dibujado por BOOLE.

Figura 13-39
Reducción de estados.



Tras elegir `Cambiar`, el BOOLE muestra el DTE de la figura 13-40.

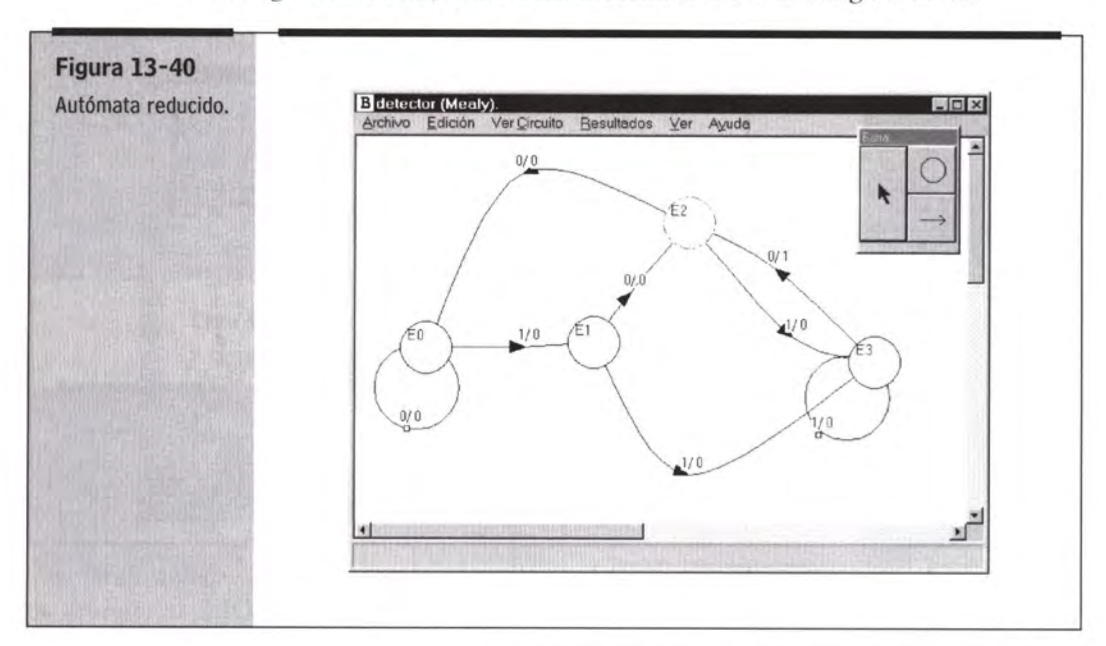


Figura 13-40
Autómata reducido.

Sólo quedan por obtener las tablas y el circuito lógico correspondiente. En la figura 13-41 vemos que las tablas son distintas de las del anterior ejercicio, como corresponde a la diferencia entre Moore y Mealy. En la figura 13-42 se ve el circuito lógico basado en biestables tipo D.

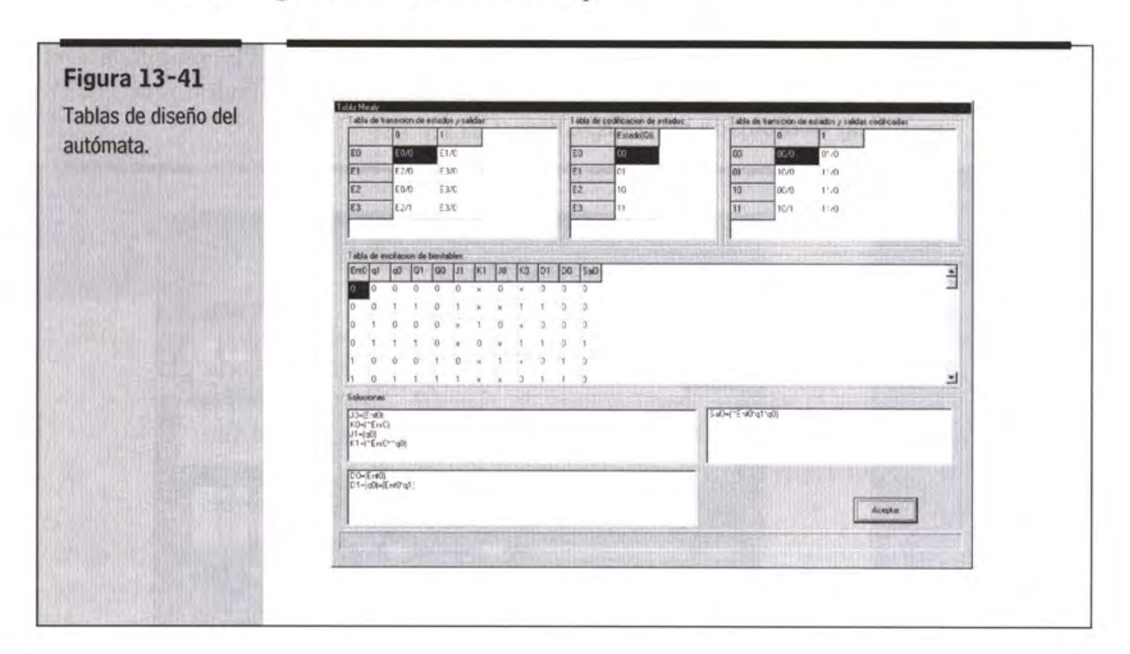


Figura 13-41
Tablas de diseño del autómata.

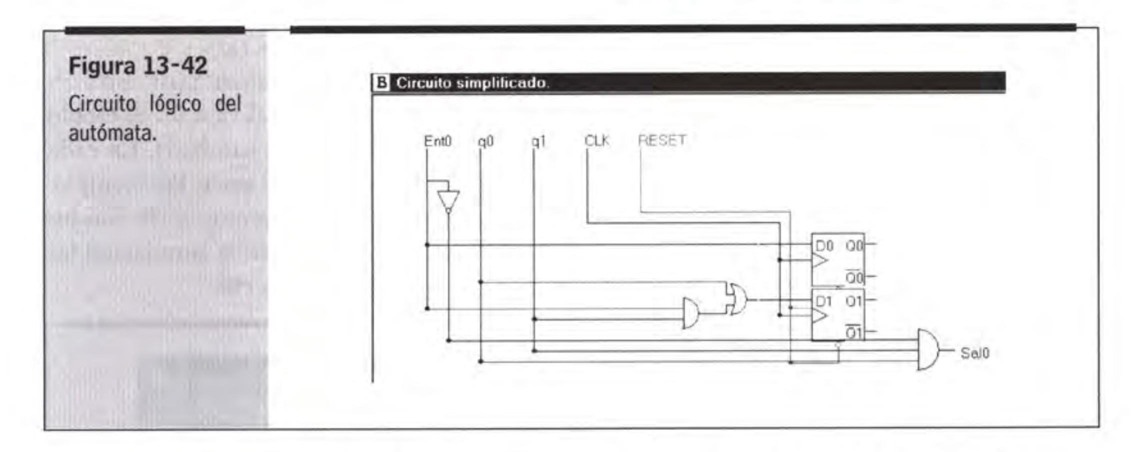


Figura 13-42
Circuito lógico del autómata.

Por último, y al igual que en los combinacionales, BOOLE permite obtener el programa en OrCAD-PLD listo para ser compilado, simulado y grabado. La figura 13-43 muestra el programa correspondiente al detector de secuencias.

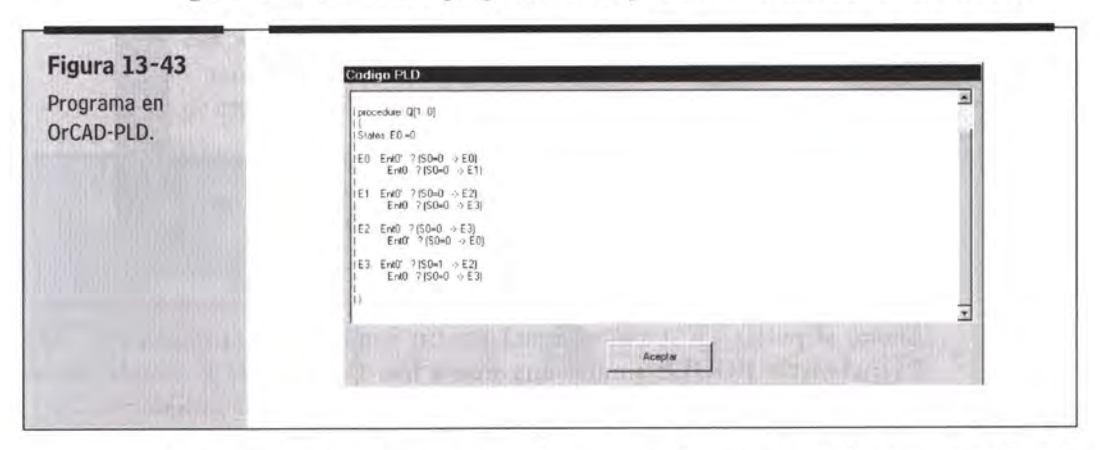


Figura 13-43
Programa en OrCAD-PLD.

Con los dos ejemplos anteriores hemos visto la potencia, didáctica y facilidad de uso del BOOLE para autómatas.

13.5.3. Simulación de autómatas

El diseño de autómatas puede llegar a complicarse, siendo en estos casos muy útil una herramienta de simulación como la que presentamos.

Una vez que el alumno ha introducido el DTE que él cree correcto puede simularlo, y así comprobar su validez. En este caso el alumno deberá preparar un juego de ensayo de forma meticulosa, contemplando el mayor número de situaciones sin llegar a eternizarse. Seguidamente cargará dicho juego de ensayo en el simulador de BOOLE, y observará si los resultados obtenidos coinciden con los esperados. BOOLE no dice si el autómata está bien, es el alumno el responsable de la simulación, enfrentando qué quiere y qué tiene.

13.5.3.1. Simulación batch

Si el alumno dentro del menú `Resultados` eligiera `Simulación Batch` pasaría a ver la figura 13-44. En ella el alumno indicará cuál va a ser el estado inicial y cuántos casos se van a simular (este número puede cambiar). En cada fila el alumno escribirá el valor de las entradas, del reloj y del reset. Por ejemplo, en la primera fila activaremos el reset, en la segunda activaremos el clk con los valores 0 y 1 en las entradas, y así sucesivamente. Destacan en la simulación las filas 5, 6 y 7, pues en ellas cambian las entradas, pero no hay clk.

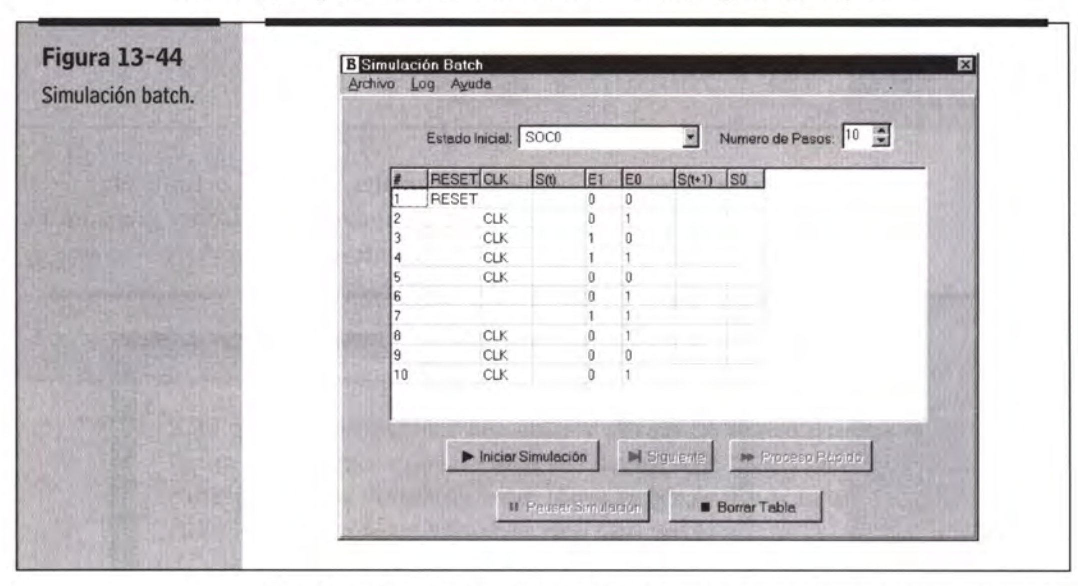


Figura 13-44
Simulación batch.

Ahora, al pulsar `Inicio Simulación` comienza la misma, y por cada `Siguiente` BOOLE simula una nueva fila. Completada la simulación se obtiene la figura 13-45, que confirma la validez del autómata introducido.

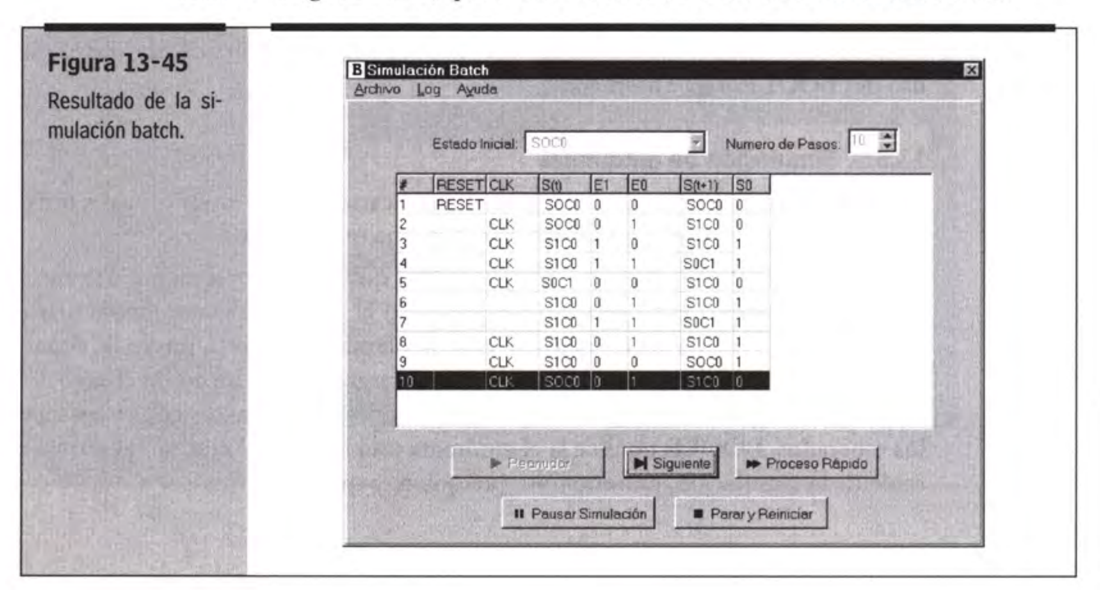


Figura 13-45
Resultado de la simulación batch.

Hay que ser cuidadoso a la hora de simular y leer la simulación, teniendo en cuenta que:

- Al llegar a una fila (en azul), ésta muestra su estado actual, su salida actual y cuál va a ser su próximo estado (cuando se haga clic y haya clk).
- Una fila no es procesada hasta no pulsar `Siguiente`, es decir, la salida que se ve en cada fila es la de la fila anterior.
- No debe verse la salida escrita en una fila como el resultado de procesar su entrada, sino como el resultado de procesar la anterior entrada.
- Esto puede parecer incómodo, pero resulta más real y potente a la hora de plantear simulaciones donde la entrada cambia entre flancos.
- Los cambios en las entradas entre flancos son vistos de distinta forma por Moore y Mealy (Mealy los ve, Moore no).

Para ver mejor lo dicho ahora, el alumno puede tener a la vista simultáneamente la pantalla de simulación y la de captura del DTE. En esta última puede ver gráficamente la simulación, por ejmplo, la figura 13-46 muestra lo dicho para esa segunda fila, viéndose cuál es el estado actual, cuál es la transición que ahora está activa (preparada para cuando se haga clic) y cuál será el próximo estado. Cuando el alumno/profesor practique con la simulación entenderá las ventajas de ésta, aunque resulte algo compleja a primera vista

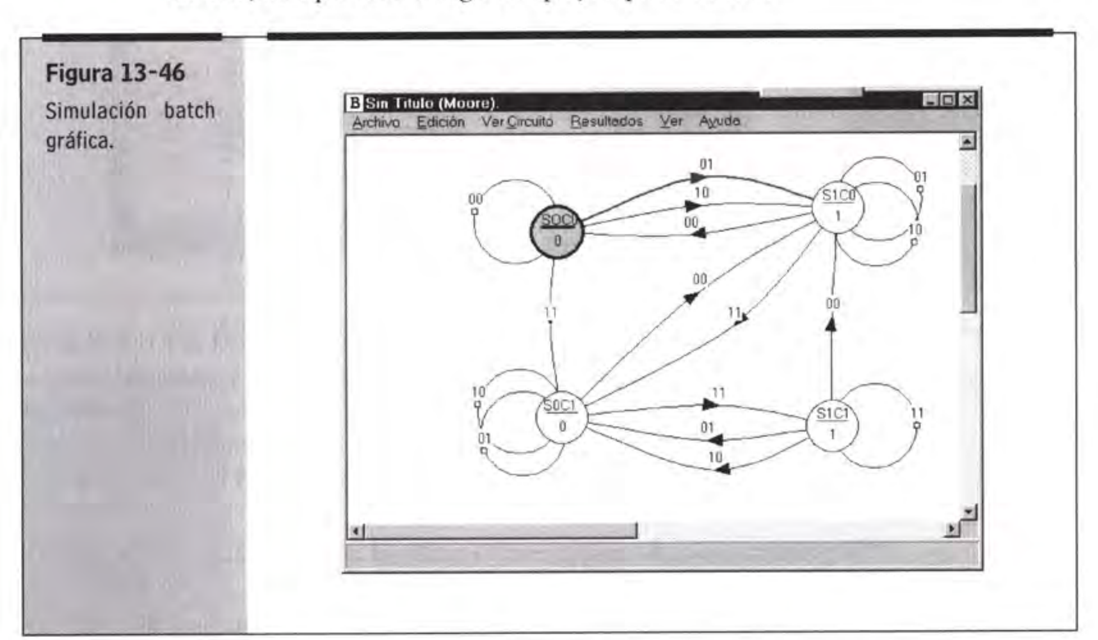


Figura 13-46 Simulación batch gráfica.

Si observamos con calma las filas 5, 6 y 7 vemos que sólo es procesada la fila 5 (cuyo resultado aparece en la fila 6).

Además de lo visto, BOOLE permite:

- Guardar y cargar una simulación. En el menú `Archivo`.

- Mostrar la simulación en un fichero de texto para imprimirlo o copiarlo en un documento del tipo que se quiera (muy cómodo para documentar ejercicios).

13.5.3.2. Simulación interactiva

BOOLE permite simular de modo interactivo. En este caso el proceso es más flexible para el alumno; él no tiene por qué preparar antes la secuencia de entrada, sino que la puede ir improvisando sobre la marcha. La figura 13-47 muestra lo que el alumno ve al elegir `Simulación Interactiva` en el menú `Resultados`.

En la pantalla se ven varios elementos que hay que manejar con cuidado. Por ejemplo, lo primero que hará el alumno será pulsar `Reset`, luego escribirá una entrada y pulsará `Aplicar`, con lo que la entrada pasa al DTE (lo que se ve en la parte gráfica de la pantalla) y finalmente hará clic sobre `clk` para que el autómata pase al nuevo estado (si así se desea).

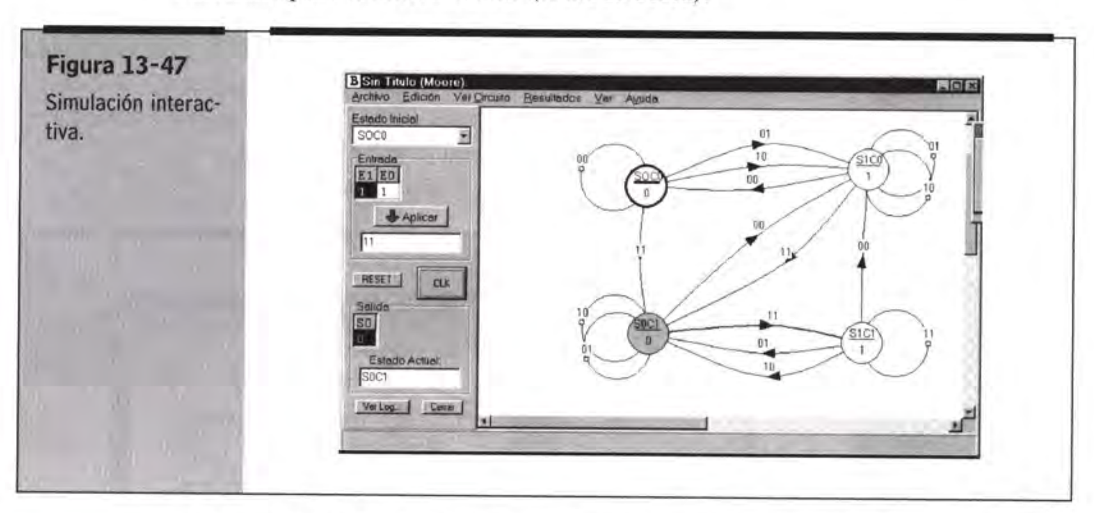
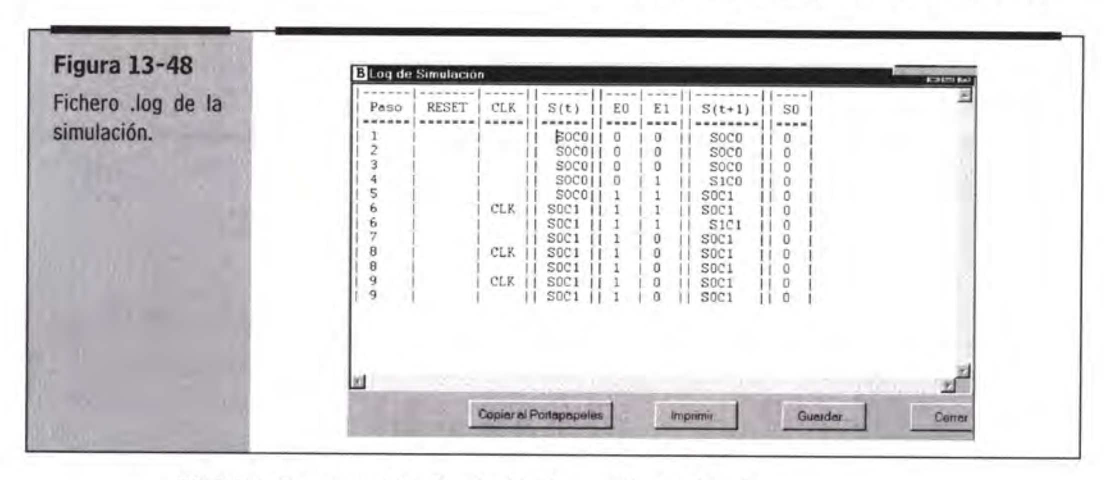


Figura 13-47
Simulación interactiva.

El desarrollo de la simulación es interactiva, pero es guardada por BOOLE en un fichero `.log`, de modo que el alumno puede recuperar y revisar lo simulado bien para estudiarlo, imprimirlo o guardarlo (activando `Ver log`). Al ver el archivo de texto `.log` sorprende el hecho de que algunas entradas tengan una fila y otras dos. El último caso coincide con la activación del clk, así si nos fijamos en las filas 6 de la figura 13-48 resulta que:

- la primera muestra en qué estado estaba el autómata al recibir la entrada (antes del clk),
- mientras que la segunda fila muestra en qué estado queda el autómata (y su nueva salida) al procesar dicha entrada.

Es decir, y siendo coherente con lo ya dicho, el resultado de un clk se ve en la fila siguiente. Esta duplicidad permite un análisis más detallado, teniendo en cuenta que si estorbaran siempre podrán ser borrados en el `.log`.



Paso	RESET	CLK	S(t)	E0	E1	S(t+1)	S0
1			SOC0	0	0	SOC0	0
2			SOC0	0	0	SOC0	0
3			SOC0	0	0	SOC0	0
4			SOC0	0	1	S1C0	0
5			SOC0	1	1	SOC1	0
6		CLK	SOC1	1	1	SOC1	0
6			SOC1	1	1	S1C1	0
7			SOC1	1	0	SOC1	0
8		CLK	SOC1	1	0	SOC1	0
8			SOC1	1	0	SOC1	0
9		CLK	SOC1	1	0	SOC1	0
9			SOC1	1	0	SOC1	0

Figura 13-48
Fichero .log de la simulación.

13.5.3.3. Comparación simulación interactiva vs batch

Simular un autómata exige ordenar las entradas respecto del reloj. La pregunta es: ¿cuándo cambia la entrada, antes del flanco, después o en el mismo instante?

Muchas veces, al simular queremos que la entrada cambie con el flanco, lo que en principio no es real. Lo normal es primero cambiar la entrada y luego generar el flanco, de esta forma en el momento del flanco se procesa la entrada ya estabilizada.

Vista la situación anterior desde BOOLE hay que tener en cuenta que:

- en la simulación interactiva primero se cambia la entrada y luego se genera el clk (`Aplicar y clk`), lo que asegura que el ritmo de la simulación es controlado por el usuario,
- en la simulación batch se puede forzar que el cambio de la entrada coincida con el flanco, ahora bien, el usuario debe recordar que el efecto de dicha entrada no será visto hasta pulsar `Siguiente`.

Por ejemplo, si queremos simular la secuencia de entrada de la tabla 13-2.

Tabla 13-2
Secuencia a simular.

RESET	CLK	A	B	SUMA
RESET		0	0	0
	↑	0	1	1
	↑	1	0	1
	↑	1	1	0
	↑	0	0	1
		0	1	1 (*)
		1	1	1 (*)
	↑	0	1	1
	↑	0	0	0
	↑	0	1	1

(*) no procesadas

Simulada la anterior secuencia mediante el simulador batch se obtiene la tabla 13-3.

Tabla 13-3

Resultado de la simulación batch.

PASO	RESET	CLK	S(t)	E0	E1	S(t+1)	S0
1	RESET		S0C0	0	0	S0C0	0
2			S0C0	0	0	S0C0	0
2		CLK	S0C0	0	1	S1C0	0
3		CLK	S1C0	1	0	S1C0	1
4		CLK	S1C0	1	1	S0C1	1
5		CLK	S0C1	0	0	S1C0	0
6			S1C0	0	1	S1C0	1
7			S1C0	1	1	S0C1	1
8		CLK	S1C0	0	1	S1C0	1
9		CLK	S1C0	0	0	S0C0	1
10		CLK	S0C0	0	1	S1C0	0

Se ve que el resultado es el mismo pero retardado un flanco, hubiera hecho falta una fila más para procesar la entrada 0 1 del último clk.

La simulación interactiva de dicha secuencia es más larga, pues primero debemos cambiar la entrada y luego procesarla, que es lo más *lógico*. El resultado obtenido es la tabla 13-4.

Las salidas de la secuencia están marcadas en negrita y coinciden con la fila siguiente al clk. Dicha fila, por cierto, está numerada igual que la anterior. Por ejemplo, miremos a las dos filas número 3: la primera indica qué entrada hay en el momento del flanco y la segunda establece la salida tras ese flanco.

Lo dicho puede parecer complicar la simulación, y además hubiera sido más sencillo diseñar una simulación como la de la tabla 13-2. Si hemos optado por esta simulación es para ceder todo el control de la simulación al alumno/profesor, para que ambos puedan simular con libertad y no dirigidos, y poder analizar, por ejemplo, la diferencia entre Moore y Mealy.

Tabla 13-4

Resultado de la simulación interactiva.

PASO	RESET	CLK	S(t)	E0	E1	S(t+1)	S0
1	RESET		S0C0	0	0	S0C0	0
1			S0C0	0	0	S0C0	0
2			S0C0	0	0	S0C0	0
3		CLK	S0C0	0	0	S0C0	0
3			S0C0	0	0	S0C0	0
4			S0C0	0	1	S1C0	0
5		CLK	S1C0	0	1	S1C0	1
5			S1C0	0	1	S1C1	1
6			S1C0	1	0	S1C0	1
7		CLK	S1C0	1	0	S1C0	1
7			S1C0	1	0	S1C0	1
8			S1C0	1	1	S0C0	1
9		CLK	S0C0	1	1	S0C0	0
9			S0C0	1	1	S1C0	0
10			S0C0	0	0	S1C0	0
11		CLK	S1C0	0	0	S1C0	0
11			S1C0	0	0	S0C0	1
12			S1C0	0	1	S1C1	1
13			S1C0	1	1	S0C0	1
14			S1C0	0	1	S1C0	1
15		CLK	S1C0	0	1	S1C1	1
15			S1C0	0	1	S1C0	1
16			S1C0	0	0	S0C0	1
17		CLK	S0C0	0	0	S0C0	0
17			S0C0	0	0	S0C0	0
18			S0C0	0	1	S1C0	0
19		CLK	S1C0	0	1	S1C0	1
19			S1C0	0	1	S1C0	1

13.6. Comentarios

El entorno BOOLE ha sido diseñado y programado durante cinco años por sus autores, Jesús Sanz Martínez y Javier García Zubía, y por alumnos de la Facultad de Informática de la Universidad de Deusto (ESIDE), destacando entre estos últimos Borja Sotomayor. Se han utilizado la tecnología orientada a objetos, el lenguaje de programación C++ y la librería STL.

El resultado final tiene en este momento más de 20.000 líneas de código (excluyendo comentarios y líneas en blanco), lo que da una idea de la complejidad del entorno. BOOLE ha sido presentado en cuatro congresos nacionales e internacionales, y ha recibido el "Premio al Mejor Equipo Software" en el IV Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica, TAEE 2000, celebrado en Barcelona en septiembre del año 2000.

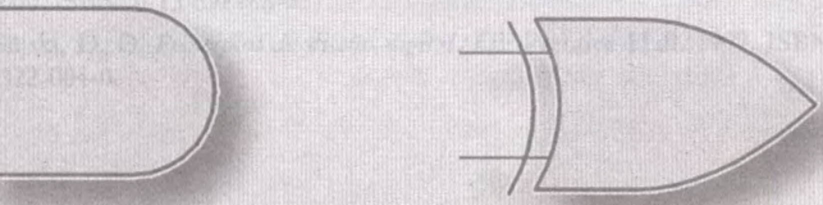
El entorno ha sido probado insistentemente por sus autores, sus alumnos (700 alumnos al año de ingeniería informática e industrial), otros profesores (Alfonso Barba y José Antonio Aranguren) y por otros colaboradores externos (gracias especialmente a Mariano Barrón y Javier Martínez), pero esto no es óbice para que no aparezcan nuevos errores y comentarios.

Agradeceríamos a todos aquellos que encuentren un error o tengan cualquier comentario al entorno (positivo o negativo) lo envíen a la dirección: *zubia@eside.deusto.es*, lo que nos permitirá mejorar BOOLE.

Pensamos que una vez creado el entorno es responsabilidad de todos los que usemos BOOLE el mantenerlo en condiciones, para que todo este esfuerzo no sea baldío.

BIBLIOTECA
CADIZ
PUBLICA

BIBLIOGRAFÍA



Bibliografía

A continuación enumeramos los libros que a nuestro parecer pueden resultar interesantes para el lector.

Angulo, J. M. *Introducción a los computadores*. 2ª edición. Ed. Paraninfo. 1995. ISBN: 84-283-2213-9.

Angulo, J. M. *Electrónica digital moderna*. Ed. Paraninfo. 1993. ISBN: 84-283-2038-1.

Baena, C. et al. *Problemas de circuitos y sistemas digitales*. Ed. Mc Graw Hill. 1997. ISBN: 84-481-0966-X.

Biswas, N. N. *Logic design theory*. Prentice Hall International Editions. 1993. ISBN: 0-13-010695-X.

Blanco Flores, F. y Olvera Peralta, S. *Electrónica digital y microprogramable*. Ed. Paraninfo. 1998. ISBN: 84-283-2482-4.

Boole, G. *An investigation of the laws of thought*. Dover Publications. ISBN: 0-486-60028-9.

Bracho del Pino, S. *La ingeniería microelectrónica ante el cambio de milenio*. Servicio de publicaciones de la Universidad de Cantabria. 1999. ISBN: 84-8102-236-5.

Brayton, R. K. y otros. *Logic minimization algorithms for VLSI synthesis*. Kluwer academic publishers. 6ª reimpresión. 1990. ISBN: 0-89838-164-9.

Brown, S. y Vranesic, Z. *Fundamentals of logic design with VHDL design*. Ed. Mc Graw Hill. 2000. ISBN: 0-07-012591-0.

Cembranos Nistal, F. J. *Sistemas de control secuencial*. Ed. Paraninfo. 1998. ISBN: 84-283-2508-1.

Delgado García, A. E. et al. *Problemas de electrónica digital*. Ed. Sanz y Torres. 1995. ISBN: 84-88667-11-6.

Delgado, A. E. y otros. *Problemas de electrónica digital*. Ed. Sanz y Torres. 1999. ISBN: 84-88667-46-9.

Deschamps, J. P. y Angulo, J. M. *Diseño de sistemas digitales*. 2ª edición. Ed. Paraninfo. 1992. ISBN: 84-283-1695-3.

Dokter, F. y Steinhauer, J. *Electrónica digital. Fundamentos teóricos y técnica de circuitos*. Ed. Paraninfo. 1977. ISBN: 84-283-0884-5.

Ercegovac, M.; Lang, T. y Moreno, J. H. *Introduction to digital systems*. Ed. Jon Wiley & Sons. 1999. ISBN: 0-471-57299-8.

Floyd, T. L. *Fundamentos de sistemas digitales*. 7ª edición. Ed. Prentice Hall. 2000. ISBN: 0-13-398488-5.

Gajski, D. D. *Principios de diseño digital*. Ed. Prentice Hall. 1997. ISBN: 84-8322-004-0.

García Sánchez, J. E. y otros. *Circuitos y sistemas digitales*. Ed. Tébar Flores. 1992. ISBN: 84-7360-125-4.

García Zubía, J. *Ejercicios básicos de electrónica digital*. Ed. Universidad de Deusto. 2000. ISBN: 84-7485-701-5.

Gardner, M. *Máquinas y diagramas lógicos*. Alianza Editorial. 1985. ISBN: 84-206-0091-1.

Gascón de Toro y otros. *Problemas prácticos de diseño lógico*. Ed. Paraninfo. 1990. ISBN: 84-283-1731-3.

Gassner, E. *Tablas de equivalencia. Diccionario TTL en 6 idiomas*. Ed. Alfaomega. 1995. ISBN: 970-15-0103-9.

Gil Padilla, A. J. et al. *Electrónica digital y microprogramable*. Ed. Mc Graw Hill. 1999. ISBN: 84-481-0927-9.

Hayes, J. P. *Introducition to digital logic design*. Ed. Addison Wesley. 1993. ISBN: 0-201-15461-7.

Hermosa Donate, A. *Electrónica digital fundamental*. 2ª edición. Ed. Marcombo. 1997. ISBN: 84-267-1133-2.

Hermosa Donate, A. *Técnicas electrónicas digitales*. Ed. Marcombo. 1997. ISBN: 84-267-1100-6.

Hill, F. J. y Peterson, G. R. *Teoría de conmutación y diseño lógico*. Noriega Editores. LIMUSA. 1997. ISBN: 968-18-0551-8.

Katz, R. H. *Contemporary logic design*. Ed. Benjamin Cummings. 1994. ISBN: 0-8053-2703-7.

Mandado, E. *Sistemas electrónicos digitales*. 8ª edición. Ed. Marcombo. 1998. ISBN: 84-267-1170-7.

Mano, M.M. y Kime, Ch. R. *Fundamentos de diseño lógico y computadoras*. Ed. Prentice Hall. 1998. ISBN: 0-13-182098-2.

Marston, R. M. *Circuitos digitales CMOS*. Ed. Paraninfo. 1996. ISBN: 84-283-2266-X.

Marston, R. M. *Circuitos digitales TTL*. Ed. Paraninfo. 1996. ISBN: 84-283-2266-X.

Mazo Quintas y otros. *Circuitos electrónicos digitales*. Ed. Universidad de Alcalá. 1995. ISBN: 84-8138-960-9.

McCluskey, E. J. *Logic design principles with emphasis on testable semicustom circuits*. Ed. Prentice Hall. 1986. ISBN: 0-13-539784-7.

Meinel, Ch. y Theobald, T. *Algorithms and data structures in VLSI design*. Ed. Springer. 1998. ISBN: 3-540-64486-5.

Mira, J. J. et al. *Electrónica digital*. Ed. Sanz y Torres. 1993. ISBN: 84-88667-04-3.

Nelson, V. P. y otros. *Análisis y diseño de circuitos lógicos digitales*. Ed. Prentice Hall. 1996. ISBN: 0-13-463894-8.

Newton, A.R. Editor. *Logic synthesis for integrated circuit design*. IEEE Press. New York. 1987. ISBN: 0-87492-236-X.

Ojeda Cherta, F. *Problemas de diseño de automatismos electrónico-eléctricos y electrónico-neumáticos*. Ed. Paraninfo. 1996. ISBN: 84-283-2270-8.

Ojeda Cherta, F. *Problemas de electrónica digital*. Ed. Paraninfo. 1994. ISBN: 84-283-2133-7.

Padilla, I. *Ejercicios de electrónica digital*. Servicio de publicaciones de E.T.S. de Ingenieros de Telecomunicación. 1989. ISBN: 84-7402-194-4.

Ramírez, J. *20 montajes industriales con circuitos integrados*. Ed. CEAC. 1987. ISBN: 84-329-8032-3.

Sanz, R. *Curso práctico de electrónica digital*. Ed. Rus. 1991. ISBN: 84-87723-01-2.

Tevernier, Ch. *Circuitos lógicos programables*. Ed. Paraninfo. 1994. ISBN: 84-283-2114-0.

Tocci, R. J. *Digital systems. Principles and applications*. 5th edition. Ed. Prentice Hall. 1991. ISBN: 0-13-213133-1.

Tokheim, R. L. *Principios digitales*. 2ª edición. Ed. Mc Graw Hill. 1991. ISBN: 84-7615-509-3.

Velasco Ballano, J. y Otero Arias, J. *Problemas de sistemas electrónicos digitales*. Ed. Paraninfo. 1996. ISBN: 84-283-2231-7.

Wakerly, J. F. *Digital design. Principles and practices*. 3rd edition. Ed. Prentice Hall. 2000. ISBN: 0-13-212838-1.

Wilkinson, B. *The essence of digital design*. Ed. Prentice Hall. 1998. 0-13-570110-4.

Yarbrough, J. M. *Digital logic. Applications and design*. PWS publishing co. ITP. 1997. ISBN: 0-314-06675-6.

SISTEMAS DIGITALES

y Tecnología de Computadores

La presente obra es fruto del trabajo y la experiencia de sus autores en la divulgación y enseñanza de la Tecnología Digital a los alumnos del primer curso de diversas especialidades de ingeniería en la Universidad de Deusto. Intenta servir como una eficaz herramienta para la comprensión y estudio de la Teoría de Sistemas Digitales y de los cimientos de la Tecnología de Computadores.

El libro incluye un CD con la última y más completa versión del programa BOOLE-DEUSTO, así como numerosa descripción técnica de los circuitos integrados más comunes. Esta aplicación viene avalada por diversos congresos nacionales e internacionales y por el "Premio al Mejor Equipo Software" en el TAEE 2000. Este programa potencia y facilita el trabajo del profesor, del alumno y del autodidacta, capturando y operando con tablas de verdad, diagramas de Veitch-Karnaugh, expresiones booleanas, autómatas de Moore y Mealy, circuitos lógicos, etc., convirtiéndose en lo que algunos profesionales llaman *la calculadora booleana.*

Los autores del libro, D. José Mª Angulo Usategui y D. Javier García Zubía, pertenecen al departamento de Arquitectura de Computadores y Electrónica Básica de la Facultad de Ingeniería de la Universidad de Deusto. El primero de ellos es catedrático de Arquitectura de Computadores y director del departamento, además es autor de más de cuarenta obras técnicas en esta editorial. Por su parte, Javier García Zubía, que ideó y desarrolló el programa BOOLE-DEUSTO, es profesor de Electrónica Digital y Tecnología de Computadores. En esta obra el lector tiene a su ... cia, originalidad y claridad ... os autores para dominar ... lectrónica Digital.

ISBN 84-9732-042-5

